A Simple Guide for Machine Learning Application in Logistics

Prasanna Iyer [prasanna.iyer@cnhind.com]

Table of Contents

Overview of Transit Time

The purpose of this article is to describe a practical approach for analyzing transit times within a logistics network and subsequently, predicting the transit time and/or on-time delivery of a shipment. Each section contains general approach, references and finally, contextual notes based on experience. General approach can be viewed as an introduction while the references provide resources for further in-depth study.

Transit time is a very prominent topic in Transportation Logistics. Transit time is defined as the gap (in days or hours) between the pickup and delivery of a shipment. There are different parameters that influence the variance in transit time. Intuitively, as the transit time increases, the amount of uncertainty also increases. This is not always a function of the distance as air shipments across continents have a longer distance, but shorter transit time compared to domestic shipments.

In general, a standard transit time expectation can be defined at varying levels of granularity. If a shipment is on-time is determined by comparing the actual transit time versus the standard transit time. Logistics practitioners know that customers appreciate consistency and with similar level of interest, expect reliability in expected delivery times.

Ontime delivery performance (OTP) is beneficial as a reactive measurement to address structural issues with service. In day-to-day interactions with the customers, ability to provide reliable ETAs is more valuable.

Data Science and specifically Machine Learning (ML) can be quite useful in predicting transit time and on-time performance. Using a ML model, it is possible to predict the ETA as well as if a shipment will be delivered on-time or late. The most famous and widely used transit time predictor is Google Maps. Similar use cases in action can be seen at Instacart & Uber.

Machine Learning for Transit Time Prediction

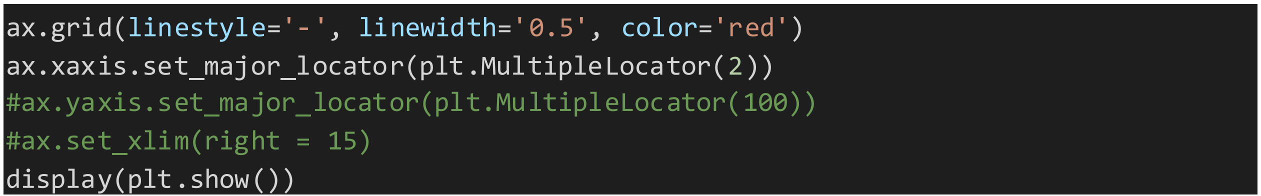
There are many excellent articles on the workflow to follow for an ML project. The main steps are Data Gathering, Exploratory Data Analysis, Feature Engineering, Model Building, Inference.

Exploratory Data Analysis

General Approach

There are two kinds of plots that are particularly useful. The first one is the KDE [Kernel Density Estimate] plot that shows the distribution of a particular variable. While a histogram shows the frequency of the different values [categorical features] or ranges [continuous features], KDE plot shows the probabilistic distribution for continuous features.

```
import seaborn as sns
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
## df_copy1 is a Dataframe containing the data for analysis
fig = plt.figure(figsize = (10,6.5))
ax = sns.kdeplot(data=df_copy1[(df_copy1['Calc_Delay']>-10) &
(df_copy1['Calc_Delay']<10)]['Calc_Delay'],cumulative = True)
```



```
ax.grid(linestyle='-', linewidth='0.5', color='red')
ax.xaxis.set_major_locator(plt.MultipleLocator(2))
#ax.yaxis.set_major_locator(plt.MultipleLocator(100))
#ax.set_xlim(right = 15)
display(plt.show())
```

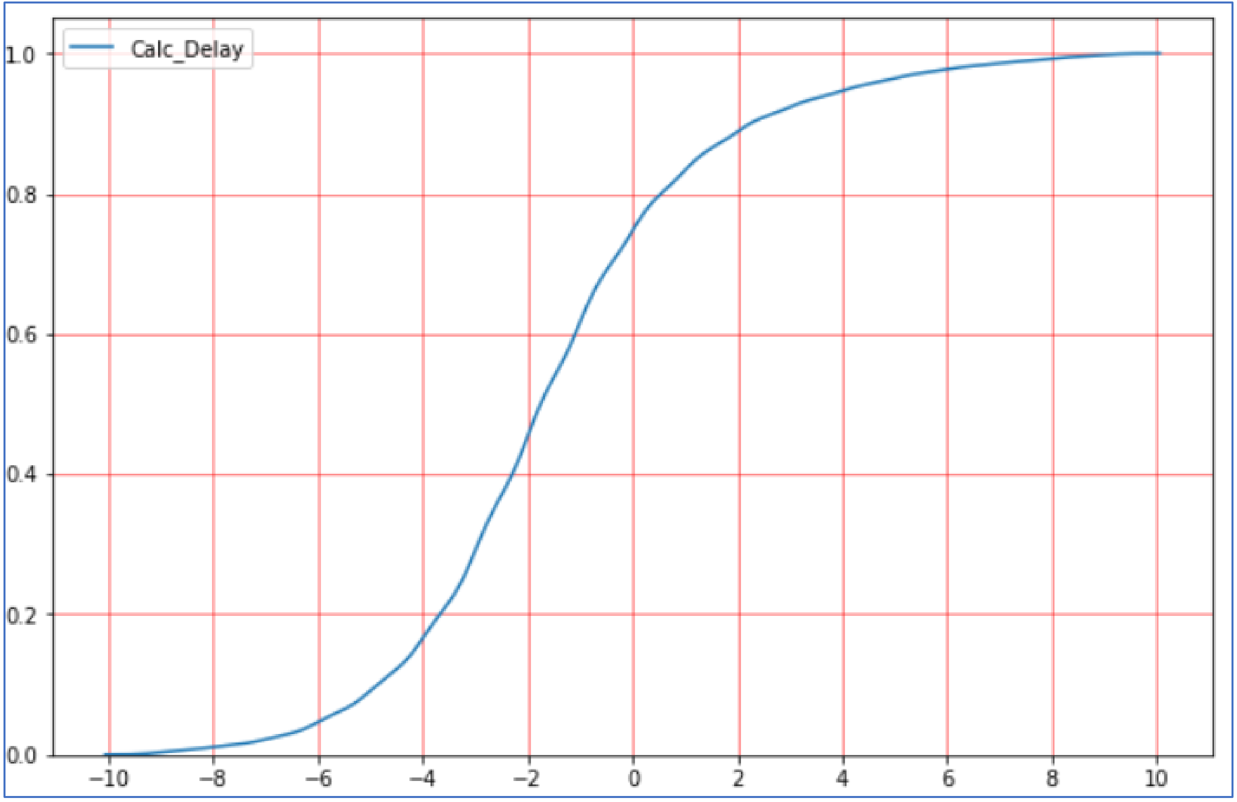


Figure 1: KDE plot example

Figure 1 shows the KDE plot for a variable called “Calc_Delay”. It is a suitable method to see the distribution of a numerical variable. KDE plot does smooth out the outliers thereby creating a visual view of the distribution.

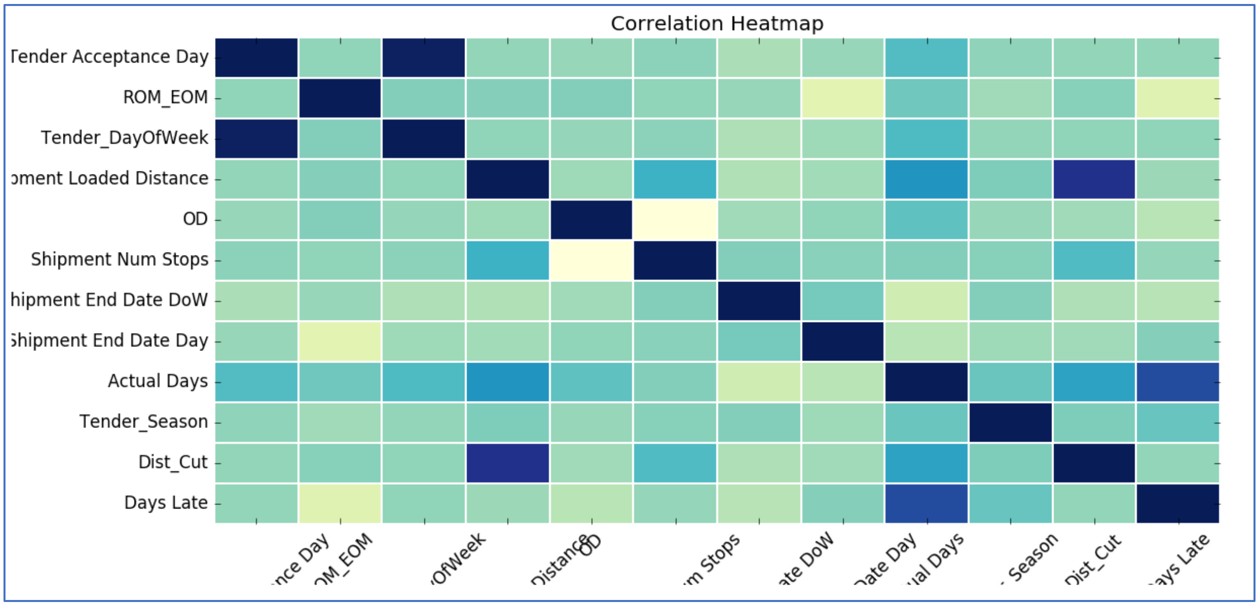


Figure 2: Correlation Heatmap

The second one is the correlation plot. Such as plot shows the correlation among all the numerical variables in dataset. This plot will show the features that are highly correlated among themselves as well as with the target variable.

```
## df_copy1 is a Dataframe containing the data for analysis
fig = plt.figure(figsize = (10,6.5))
ax = sns.heatmap(df_copy1.corr())
display(plt.show())
```

The third one is the box plot. This plot shows the distribution of the numerical features and more interestingly the outliers. This plot is one of the necessary steps before applying outlier exclusion techniques such as IQR [Inter Quartile Range].

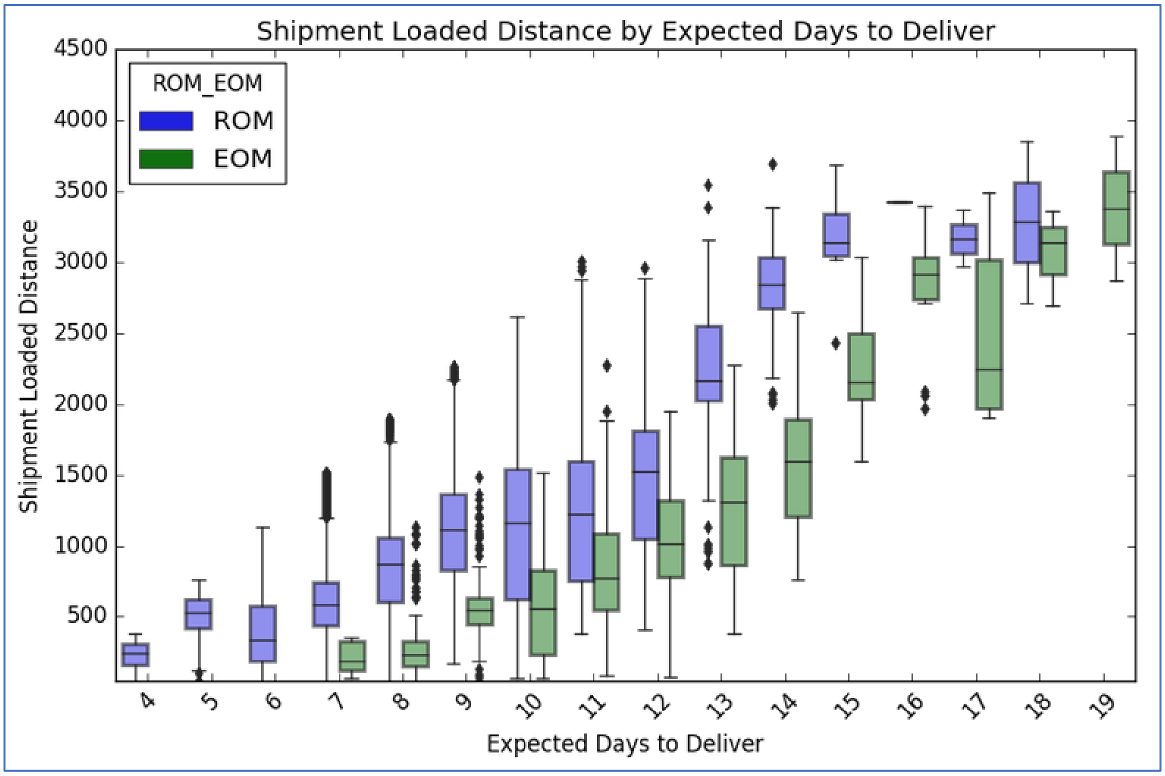


Figure 3: Box plot example

Figure 3 shows the Box plot for a numerical variable [y-axis] versus a categorical variable [x-axis]. For the numerical variable, the plot shows the minimum, median and maximum [within the box]. The outliers are shown outside the box. Box plot is a suitable option to view the range of a variable as well as the extent of outliers.

```
## df_copy1 is a Dataframe containing the data for analysis
fig = plt.figure(figsize = (10,6.5))
ax = sns.boxplot(x=df_copy1['Expected Days to Deliver'], y= df_copy1['Shipment Loaded Distance'], hue = df_copy1['ROM_EOM'])
display(plt.show())
```

References

https://jakevdp.github.io/PythonDataScienceHandbook/

https://seaborn.pydata.org/index.html

https://shopify.engineering/conducting-exploratory-data-analysis

Contextual Notes

In-depth Exploratory Data Analysis [EDA] in the early stages of a data science project is necessary. EDA offers insights into the dataset.

One of the first observations from the EDA on transit time was that the deviation from the standard transit time [different between actual and standard times] was nearly a normal curve. However, the curve was not centered on zero. Greater proportion of the on-time shipments were arriving earlier than expected. The extent of deviation for shipments arriving early increased with distance. In other words, shipments travelling longer distances had a greater likelihood of arriving earlier [75% of the shipments have distance < 1200 mi]. This effect became more prominent for over-dimensional as well as cross-border shipments. Further, exploring the relationship between on-time performance and shipment tender day, showed that the shipments tendered on Thursdays and Fridays had a greater likelihood of being late than other shipments. Based on these observations, the business rules pertaining to the calculation of the standard transit time were modified. For example, the number of additional for over-dimensional and cross border shipments as well as daily miles driven were modified.

Feature Engineering

General Approach

Categorical features must be encoded. There are standard libraries in scikit learn for encoding. The standard encoding options converting the categorical variables into ordered numerical values and one-hot encoding. There are pros and cons (as usual) pertaining to the two options.

Fastai's tabular library provides convenience pre-processing tools. From the fastai documentation:

- **Categorify** is going to take every categorical variable and make a map from integer to unique categories, then replace the values by the corresponding index.
- **FillMissing** will fill the missing values in the continuous variables by the median of existing values (or a specific value)
- **Normalize** will normalize the continuous variables (subtract the mean and divide by the std)

Date features can be converted into numerous individual features [Year, Month, Dayofweek, Dayof month, Is_month_end, Is_month_start etc.]. Fastai has a helper function "add_datepart", which can generate the additional features.

References

https://docs.fast.ai/tabular.core.html

Contextual Notes

In transit time analysis, tender date and ship date impact the transit time. Tender date is the date when the shipment was accepted by a service provider. Ship date is the date when shipment was picked up by the service provider. The volume of shipments ramp up towards the end of the month and this phenomenon is more pronounced for quarter-end and year-end. In order to capture the variance in shipment volume, additional features were added. Firstly, new features such day of week, day of month, last day of month, last week of month, holiday etc. were created. Secondly, daily volume [for the preceding 3 business days] and weekly volume [for the preceding 2 weeks] were added. The features were created both at the shipment location level as well as the service provider level.

Model Building

General Approach

Fastai's library is a very good choice for tabular data. It is supported by the course as well as numerous examples of the application. As mentioned in the previous section, the key benefits of fastai library are:

1) Out of the box pre-processing functions for feature engineering and the functions are easy to use
2) Neural network implementation provides easy-to-use functionality to create embeddings for categorical features

Decision tree based algorithms are a robust option for tabular data. Widely used and proven options are RandomForest, XGBoost and LGBM.

Most of these libraries follow the standard scikit learn API structure with the fit and score APIs. Choice of hyperparameters is an important step when using these libraries.

```
m1 = RandomForestClassifier(n_estimators = 800, max_features = 0.7,\
min_samples_leaf=4, n_jobs=-1)
m1.fit(x_train1,y_train)
m1.score(x_train1,y_train)
```

References

https://auto.gluon.ai/dev/index.html

https://www.cs.princeton.edu/courses/archive/spring07/cos424/papers/boosting-survey.pdf

https://explained.ai/gradient-boosting/index.html

The section on RandomForest and of course, the overall course “Practical Deep Learning for Coders” by Jeremy Howard is a really great reference.

Contextual Notes

As shown below, once the input dataframe is available, with few lines of codes, machine learning model can be created and trained.

```
procs = [FillMissing,Categorify,Normalize]
tab_databunch1 =
TabularDataBunch.from_df(path,inpDf,dep_var,valid_idx=val_idx1,\
    cat_names=cat_columns,procs = procs1,cont_names=cont_columns)

learner1 = tabular_learner(tab_databunch1,layers=[1000,500],\
     emb_szs=emb_sizes,metrics=rmse)
learner1.fit_one_cycle(20, 1e-2)
```

Autogluon is an interesting library for automating the pre-processing and model building phase. This library also simplifies the creation of ensembles. It is also possible to retrieve the output of the data pre-processing and this output can be used independently with pipelines for machine learning models or analysis.

```
nn_options = { 'num_epochs': 10}
#hyperparameters = {'NN': nn_options}
problemType = 'regression'
hyperparameters={'NN': {'num_epochs':20},'GBM': {'num_boost_round':1000},'CAT':
{'iterations':10000},'RF': {'n_estimators':300},\
    'XT': {'n_estimators':300},'KNN': {},'custom': ['GBM']}

hp1={'GBM': {'num_boost_round':10000},'CAT': {'iterations':10000},'RF':
{'n_estimators':500},\
    'XT': {'n_estimators':500},'KNN': {},'custom': ['GBM']}

time_limits = 2*60

predModel = task.fit(train_data = train_data,label = labelColumn,hyperparameters
= hp1,\
    output_directory = modelDir,problem_type = problemType)

predModel1 = task.fit(train_data = train_data,label = labelColumn,hyperparameters
= hp1,\
    output_directory = modelDir1,problem_type =
problemType,num_bagging_folds=5,stack_ensemble_levels=1)
```

As shown in the above code snippet, Autogluon is an efficient way to try different machine learning algorithms. It is quite helpful to review the source code as well as the documentation to understand the critical steps.

Results & Next Steps

Using an ensemble model, the predicted transit time (test data) was within 1.5 days of the actual transit time.

The results were relatively better for predicting if a shipment was going to be delivered ontime or not.

Algorithm	Accuracy on Test Data
Fastai tabular	76%
RandomForest	75%
ExtraTrees	77%
XGBoost	73%

One of the next steps is to incorporate the predictions [transit time as well as on-time delivery] into the Transportation Management System [TMS] and thereby, share the information with the internal/external customers.

In parallel, explore if the results of the two models [transit time and on-time delivery prediction] can be combined for better insights.

Finally, it would be worthwhile to study the feature importance and find ways to present the interpretation along with the prediction.