Preprocessing

May 28, 2022

1 Preparing data

Prediction of sales is the central task in this challenge. you want to predict daily sales in various stores up to 6 weeks ahead of time. This will help the company plan ahead of time.

The following steps outline the various sub tasks needed to effectively do this:

[1]:
```
# importing of libraries
import numpy as np
import pandas as pd
import warnings
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
import seaborn as sns
import os,sys
sys.path.append(os.path.abspath(os.path.join('../scripts')))
from timeseries import TimeSeries
from clean import Clean
sns.set()
warnings.simplefilter(action='ignore', category=FutureWarning)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
pd.options.mode.chained_assignment = None  # default='warn'
plt.rcParams["figure.figsize"] = (12, 8)
pd.set_option('display.max_columns', None)
```

```
2022-05-25 04:46:28.004293: W
tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load
dynamic library 'libcudart.so.11.0'; dlerror: libcudart.so.11.0: cannot open
shared object file: No such file or directory
2022-05-25 04:46:28.004422: I tensorflow/stream_executor/cuda/cudart_stub.cc:29]
Ignore above cudart dlerror if you do not have a GPU set up on your machine.
```

```
/home/martin/Documents/pharm_sales/notebooks
```

2 Preprocessing

It is important to process the data into a format where it can be fed to a machine learning model. This typically means converting all non-numeric columns to numeric, handling NaN values and generating new features from already existing features.

In our case, you have a few datetime columns to preprocess. you can extract the following from them: weekdays weekends number of days to holidays Number of days after holiday Beginning of month, mid month and ending of month (think of more features to extract), extra marks for it

As a final thing, you have to scale the data. This helps with predictions especially when using machine learning algorithms that use Euclidean distances. you can use the standard scaler in sklearn for this.

[2]:
```
store = pd.read_csv("../data/store.csv")
df = pd.read_csv("../data/train.csv")
clean_df = Clean(df)
clean_df.merge_df(store,'Store')
clean_df.save(name='../data/unclean_train.csv')
clean_df.drop_missing_values()
clean_df.fix_outliers('Sales',25000)
clean_df.remove_unnamed_cols()
clean_df.transfrom_time_series("Store","Date")
clean_df.save(name="../data/training.csv")
```

```
/tmp/ipykernel_8234/2395586539.py:2: DtypeWarning: Columns (7) have mixed types.
Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("../data/train.csv")
2022-05-25 04:46:40,391:logger:Successfully initialized clean class
2022-05-25 04:46:40,774:logger:Successfully merged the dataframe
2022-05-25 04:46:48,947:logger:Successfully saved the dataframe
2022-05-25 04:46:50,517:logger:Successfully dropped the columns with missing
values
2022-05-25 04:46:50,532:logger:Successfully stored the features
2022-05-25 04:46:50,592:logger:Successfully handled outliers
2022-05-25 04:46:50,640:logger:Successfully removed columns with head unnamed
2022-05-25 04:46:51,338:logger:Successfully transformed data to time series data
2022-05-25 04:46:57,764:logger:Successfully saved the dataframe
```

- Ok so for handling the non-numeric columns we shall check that out in the label encode function, but for the above functions what we do is we start by merging the columns from the store.csv, and the train.csv to have one dataframe, is it really necessary for having different csv's for analysis, yes! because this leads to to better modularity of the analysis.
- Then for handling the missing values, dropping them is the better option for now, this is the reason why when we have some columns that have some missing values, it really construes the data because the previous values tell a little unless acompanied with a batch of data, and that is why I preferred the dropping of the columns
- For fixing outliers if we notice with our data through the describe summary we realize that it is not in normal cases where stores will make above 25000 because in the boxplot we notice that they are forming the part of the outliers.
- Then in the transformation of the data into the time series, in our case we split the data into weekdays, weekends, day, month, week, and year. That can be see from the transform_time_series function

[3]:
```
df = pd.read_csv("../data/test.csv")
clean_df = Clean(df)
clean_df.merge_df(store,'Store')
clean_df.save(name='../data/unclean_test.csv')
clean_df.drop_missing_values()
clean_df.fix_outliers('Sales',25000)
clean_df.remove_unnamed_cols()
clean_df.transfrom_time_series("Store","Date")
clean_df.save(name="../data/testing.csv")
```

```
2022-05-25 04:46:58,001:logger:Successfully initialized clean class
2022-05-25 04:46:58,020:logger:Successfully merged the dataframe
2022-05-25 04:46:58,343:logger:Successfully saved the dataframe
2022-05-25 04:46:58,393:logger:Successfully dropped the columns with missing
values
2022-05-25 04:46:58,394:logger:Successfully stored the features
2022-05-25 04:46:58,396:logger:Successfully handled outliers
2022-05-25 04:46:58,402:logger:Successfully removed columns with head unnamed
2022-05-25 04:46:58,477:logger:Successfully transformed data to time series data
2022-05-25 04:46:58,752:logger:Successfully saved the dataframe
```

[4]:
```
train = pd.read_csv("../data/training.csv")
test = pd.read_csv("../data/testing.csv")
```

Feature Engineering

[5]:
```
daily_sales = clean_df.aggregations(train,'Store','Sales','Open','sum')
daily_customers = clean_df.aggregations(train,'Store','Customers','Open','sum')
avg_sales = clean_df.aggregations(train,'Store','Sales','Open','mean')
avg_customers = clean_df.aggregations(train,'Store','Customers','Open','mean')
train['DailySales'] = train['Store'].map(daily_sales)
train['DailyCustomers'] = train['Store'].map(daily_customers)
train['AvgSales'] = train['Store'].map(avg_sales)
train['AvgCustomers'] = train['Store'].map(avg_customers)
```

```
2022-05-25 04:46:59,588:logger:successful aggregation
2022-05-25 04:46:59,612:logger:successful aggregation
2022-05-25 04:46:59,683:logger:successful aggregation
2022-05-25 04:46:59,708:logger:successful aggregation
```

For feature engineering the columns that are necessary for this particular analysis are; - DailySales - this is an aggregation of the daily sales per store - DailyCustomer - this is an aggregation of the daily customers per store - AvgSales - this is an aggregation of the average sales per store - AvgCustomers - this is an aggregation of the average customers per store

Why are these features necessary?. It is because the weight of the information is on the sales, because we are doing sales predictions and that the relationship between the customers and the sales is a positive linear relationship, therefore it shall come in handy in the predictions

Label Encoding

[6]:
```
clean_df.label_encoding(train,test)
```

```
2022-05-25 04:46:59,750:logger:Successfully stored the features
2022-05-25 04:47:00,279:logger:Successfully encoded your categorical data
```

The encoding that we do for the categorical data features is we use the factorize method, which encodes each element of the data to a coresponding integer

Scaling Data

[7]:
```
training_data_ = train[train.columns.difference(['DayOfWeek','Day', 'Month',␣
↪'Year', 'DayOfYear','WeekOfYear'])]
testing_data_ = test[test.columns.difference(['DayOfWeek','Day', 'Month',␣
↪'Year', 'DayOfYear','WeekOfYear'])]
```

[8]:
```
train_transformation=clean_df.
↪generate_transformation(training_data_,"numeric","number")
test_transformation=clean_df.
↪generate_transformation(testing_data_,"numeric","number")
```

```
2022-05-25 04:47:00,646:logger:Successfully transformed numerical data
2022-05-25 04:47:00,664:logger:Successfully transformed numerical data
```

[9]:
```
indexes = ['DayOfWeek','Day', 'Month', 'Year', 'DayOfYear','WeekOfYear']
train_transformed = pd.DataFrame(train_transformation,columns=train.columns.
↪difference(indexes))
test_transformed = pd.DataFrame(test_transformation,columns=test.columns.
↪difference(indexes))
train_index = train[indexes]
test_index = test[indexes]
train = pd.concat([train_index,train_transformed],axis=1)
test = pd.concat([test_index,test_transformed],axis=1)
```

For the transformations of our data we realize that the minmaxscaler because we want to ensure that when doing the backpropagation it shall be more stable,also it shall help our code to run faster when placing it.

[10]:
```
train.sort_values(["Year","Month","Day"], ascending=False ,ignore_index=True,␣
↪inplace=True)
test.sort_values(["Year","Month","Day"], ascending=False ,ignore_index=True,␣
↪inplace=True)
```

[11]:
```
train.to_csv("../data/cleaned_train.csv",index=False)
test.to_csv("../data/cleaned_test.csv",index=False)
```

[ ]:
```

```