Project Report

1. Introduction

1.1 Overview

This project is part of IBM Hack Challenge 2020. The goal of this project is to help in the optimised warehouse management for perishable goods for a food delivery company, and to that extent offers a demand prediction machine learning model.

1.2 Purpose

The purpose of this project is to help the food delivery companies with the restocking of perishable food supplies, which if overstocked would rot, or otherwise lead to frequent restocking, thereby incurring a lot of unwarranted expenses.

2. Literature Survey

2.1 Existing Problem

A common problem faced by companies across all sectors is accurate demand supply prediction. Inaccurate predictions would incur additional expenses on part of the company being manifested in the form of excess transportation costs, or a loss in overall sales.

With respect to a company in the food sector, demand prediction is required for appropriate stocking of supplies, which are mainly perishable. Understocking of the supplies in the warehouse, would lead to the materials being used up, and the company would have to bear excess tranportation cost for restocking of the warehouse before the scheduled restocking time. Overstocking on supplies, would lead to wastage, as well as the supplies would rot, incurring a loss for the company.

Therefore, accurate demand prediction is necessary for companies to avoid additional expenditures.

2.2 Proposed Solution

Given the data for the past X weeks, say with columns suggesting the meal id, the type of cuisine, and number of orders, prices of the meals, discounts offered, fulfilment centers, we propose the following:

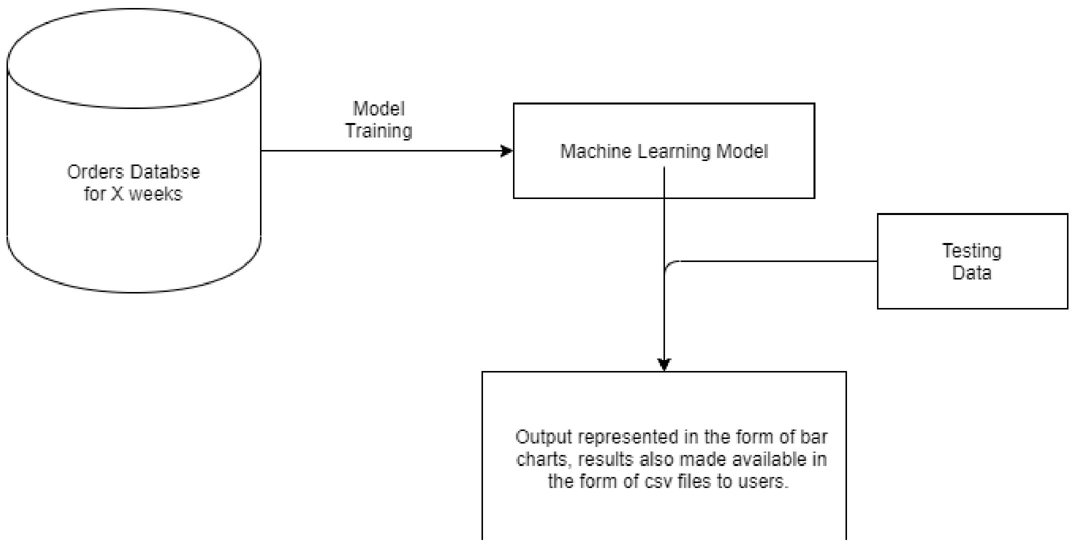
We have segregated the data according to :

- cuisine,
- meal id,
- fulfillment center id, and
- category type,

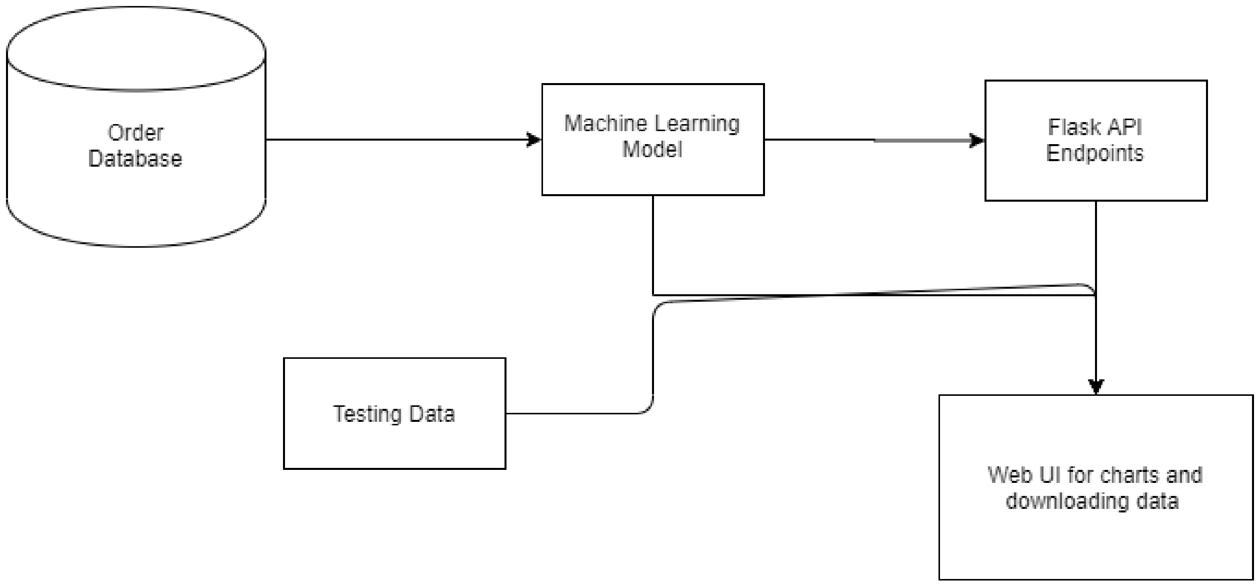
We predict the demand of various meals according to the above caegories for the next Y weeks, using various regressors such as RandomForest, XGBooster, CatBooster, to name a few.
All the above predictions, are represented in the form of charts to visually aid the user, in understanding the number of orders expected in the coming weeks.

3. Thereotical Analysis

3.1 Block Diagram


Orders Databse for X weeks
Model Training
Machine Learning Model
Testing Data
Output represented in the form of bar charts, results also made available in the form of csv files to users.

3.2 Software Designing



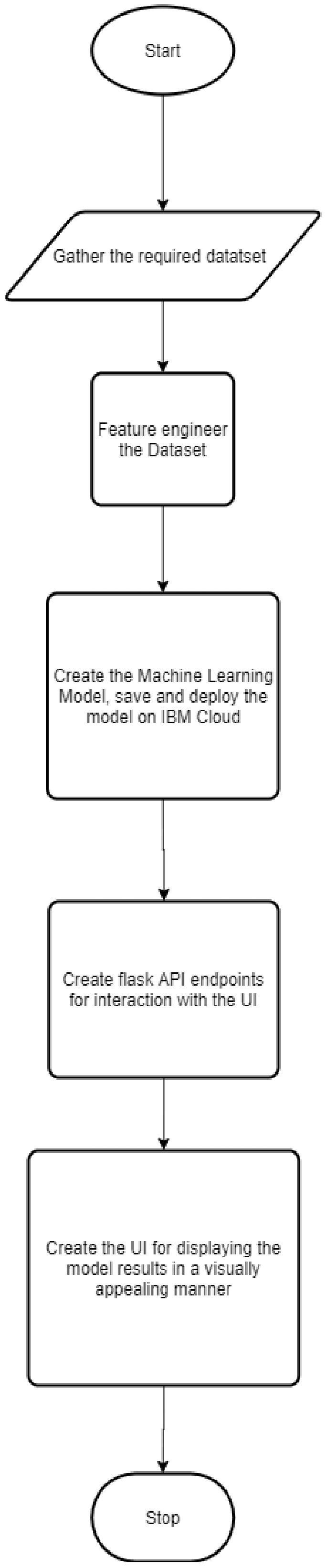
4. Experimental Investigations

For the Machine Learning Model, we had to feature engineer the dataset given in the competition reference link. We calculated the discounts given for each meal id, one hot encoded categorical values, also calculated the week sine and cosine values, to detect any cyclic rise in the number of orders, for instance, number of orders are usually higher in the weekends as compared to the weekdays. We also changed our error/loss function to root mean square log error for better performance on such a large datatset (450000*24).

We had used AutoAi models in the beginning, but they were taking approximately 10 hours to train, also the lite plan usage was being exceeded frequently, so we switched to training our model offline and deploying the same on ibm cloud.

For that purpose, we compared various models, and found the performance of XGBRegressor the best, and consequently deployed on ibm cloud.

5. Flowchart


Start
Gather the required datatset
Feature engineer the Dataset
Create the Machine Learning Model, save and deploy the model on IBM Cloud
Create flask API endpoints for interaction with the UI
Create the UI for displaying the model results in a visually appealing manner
Stop

6. Result

We have used XGBRegressor as our machine learning model, for forecasting demand predictions. The results have been communicated to the web application with the help of Flask API endpoints.

The following are the list of endpoints created:

/predict/test: For testing a user input

/predict: For batch testing of the model against a test dataset

/meal/predictions: For getting the predictions of orders per meal_id of batch testing in json format.

/meal/train/predictions: For getting the number of orders per meal_id in json format

/meal/predictions/download: For downloading the predictions of orders per meal_id of batch testing

/center/predictions: For getting the predictions of orders per center_id of batch testing in json format.

/center/train/predictions: For getting the number of orders per center_id in json format

/center/predictions/download: For downloading the predictions of orders per meal_id of batch testing

/category/predictions: For getting the predictions of orders per category of batch testing in json format.

/category/train/predictions: For getting the number of orders per category in json format

/category/predictions/download: For downloading the predictions of orders per category of batch testing

/cuisine/predictions: For getting the predictions of orders per cuisine of batch testing in json format.

/cuisine/train/predictions: For getting the number of orders per cuisine in json format

/cuisine/predictions/download: For downloading the predictions of orders per cuisine of batch testing

The data from the above API enpdoints have been represented in the form of charts in the web application, and also have been made available or downloading.

The charts in the web application have four categories: meal_id, center_id, category, cuisine. Corresponding to each category charts, both training and testing, have been showcased for the user's convinience.
In addition the home page sports an interactive form where the user can input the data, and see the predicted orders corresponding to his input.

7. Advantages and Disadvantages

The advantage of using the above model, would help the company/companies from incurring additional costs due to faulty demand predictions.
The disadvantage on the other hand as of now, is that the platform has been designed for one particular user. In the event of expansion of userbase, separate user accounts have to be maintained for usage, also separate models traine to prevent any bottleneck arising on the performance of the model.

8. Applications

The above model developed can be further extended to encompass any company that needs a demand prediction to be done for optimising their additional/miscellaneous expenditures, be it a food delivery company, or a company in sales market,etc.

9. Conclusion

Through this project we have tried to create a model that can be an aid to a food delivery company in optimising their stocks of perishable materials in warehouses, thereby helping them save additional expenses.

10. Future Scope

We can scale the platform to cater to multiple users, by creating separate user accounts, also deploying separate models with each account,maintaining a database with respect to each account as well.
Also given the costs required in transportation of goods, and the purchase expenses, as an added feature, the weekly savings due to the accurate predictions of the model can also be showcased.

11. Bibliography

https://www.altexsoft.com/blog/demand-forecasting-methods-using-machine-learning/
https://www.kaggle.com/shashkhr25/food-demand-forecasting-challange/data
https://github.com/NishantBhavsar/Genpact-ML-hackathon

APPENDIX

```
import pandas as pd
import numpy as np

fulfillment=pd.read_csv('/content/drive/My Drive/Colab
Notebooks/92422_214915_bundle_archive/fulfilment_center_info.csv')
fulfillment.head()

meal_info=pd.read_csv('/content/drive/My Drive/Colab
Notebooks/92422_214915_bundle_archive/meal_info.csv')
meal_info.head()
```

```
train=pd.read_csv('/content/drive/My Drive/Colab
Notebooks/92422_214915_bundle_archive/train.csv')
train.head()

test=pd.read_csv('/content/drive/My Drive/Colab
Notebooks/92422_214915_bundle_archive/test.csv')
test.head()

ful_merge=pd.merge(train, fulfillment,

                how="left",

                left_on='center_id',

                right_on='center_id')

ful_merge.head()

meal_merge=pd.merge(train,meal_info,how='left',left_on='meal_id',right_on='meal_id')
meal_merge.head()

full_merge=pd.merge(train,fulfillment,how='left',left_on='center_id',right_on='center_id')
full_merge=pd.merge(full_merge,meal_info,how='left',left_on='meal_id',right_on='meal_id
')
full_merge.head()

full_test=pd.merge(test,fulfillment,how='left',left_on='center_id',right_on='center_id')
full_test=pd.merge(full_test,meal_info,how='left',left_on='meal_id',right_on='meal_id')
full_test['discount_per']=((full_test['base_price']-full_test['checkout_price'])/full_test['base
_price'])*100
full_test.head()

full_merge['discount_per']=((full_merge['base_price']-full_merge['checkout_price'])/full_m
erge['base_price'])*100
full_merge.head()
```

```
from sklearn import preprocessing


from xgboost import XGBRegressor

from sklearn.model_selection import train_test_split

label_encode_columns = ['center_id',

                       'meal_id',

                       'city_code',

                       'region_code',


                       'center_type',

                       'category',

                       'cuisine']



le = preprocessing.LabelEncoder()



for col in label_encode_columns:

    le.fit(full_merge[col])

    full_merge[col + '_encoded'] = le.transform(full_merge[col])

    full_test[col + '_encoded'] = le.transform(full_test[col])

import numpy as np
```

```
full_merge['week_sin'] = np.sin(2 * np.pi * full_merge['week'] / 52.143)

full_merge['week_cos'] = np.cos(2 * np.pi * full_merge['week'] / 52.143)

full_test['week_sin'] = np.sin(2 * np.pi * full_test['week'] / 52.143)

full_test['week_cos'] = np.cos(2 * np.pi * full_test['week'] / 52.143)

columns_to_train =
['week','week_sin','week_cos','checkout_price','base_price','discount_per','emailer_for_prom
otion','homepage_featured','city_code',

'region_code','center_type_encoded','op_area','category_encoded','cuisine_encoded','cente
r_id_encoded','meal_id_encoded']

categorical_columns = ['emailer_for_promotion','homepage_featured','city_code',
'region_code','center_type_encoded','category_encoded','cuisine_encoded',
                'center_id_encoded','meal_id_encoded']

numerical_columns = [col for col in columns_to_train if col not in categorical_columns]


# Log transform the target variable - num_orders.

full_merge['num_orders_log1p'] = np.log1p(full_merge['num_orders'])

# Train-Test split.

X = full_merge[categorical_columns + numerical_columns]

y = full_merge['num_orders_log1p']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.02, shuffle=False)

print(X_train.keys())

estimator = XGBRegressor(learning_rate=0.003,n_estimators=40000,silent=False)
```

```
fit_params = {'early_stopping_rounds': 1000}

estimator.fit(X_train, y_train)

# Get predictions on the test set and prepare submission file.

# X = full_test[categorical_columns + numerical_columns]

import pickle
# loaded_model = pickle.load(open('./drive/My Drive/Colab
Notebooks/finalized_model.sav', 'rb'))
result = estimator.score(X_test, y_test)
print(result)

X = full_test[categorical_columns + numerical_columns]

pred = estimator.predict(X)
pred = np.expm1(pred)

pred

import pickle

loaded_model = pickle.load(open('./drive/My Drive/Colab
Notebooks/finalized_model.pkl', 'rb'))

filename='finalised_model.pkl'
pickle.dump(loaded_model,open(filename,"wb"))

!tar -zcvf prediction-model.tar.gz finalised_model.pkl

!pip install watson-machine-learning-client

from watson_machine_learning_client import WatsonMachineLearningAPIClient

wml_credentials = {
    "apikey": "###",
    "instance_id": "###",
    "url": "https://us-south.ml.cloud.ibm.com",
}

client = WatsonMachineLearningAPIClient(wml_credentials)
client.version

model_metadata = {
  client.repository.ModelMetaNames.NAME: "XGB",
  client.repository.ModelMetaNames.FRAMEWORK_NAME: 'scikit-learn',
  client.repository.ModelMetaNames.FRAMEWORK_VERSION: '0.20',
  client.repository.ModelMetaNames.RUNTIME_NAME: 'python',
  client.repository.ModelMetaNames.RUNTIME_VERSION: '3.6'
}

model_details = client.repository.store_model(model='prediction-model.tar.gz',
meta_props=model_metadata)

model_uid = client.repository.get_model_uid(model_details)

print( model_uid )

model_id = model_details["metadata"]["guid"]
model_deployment_details = client.deployments.create( artifact_uid=model_id,
name="XGB" )

scoring_endpoint = client.deployments.get_scoring_url( model_deployment_details )
print( "scoring_endpoint: ", scoring_endpoint )

full_test.shape

idd=[]
for i in range(full_test.shape[0]):
  idd.append(full_test['id'][i])

prediction=pd.DataFrame({'id':idd,'num_orders':pred})
```

```
prediction.to_csv('/content/drive/My Drive/Colab Notebooks/results.csv',index=False)

##plotting graphs

import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['agg.path.chunksize'] = 10000

l=list(full_merge.meal_id.unique())
print(len(l))

meal_group=full_merge.groupby('meal_id')
order_sort=meal_group['num_orders'].agg(np.sum)
order=list(meal_group['num_orders'].agg(np.sum))
print(len(order))

import seaborn as sns
plt.figure(figsize=(20,10))
plt.xlabel('Meal_ID')
plt.ylabel('No. of orders')
ax=sns.barplot(x=l,y=order)
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())

center=list(full_merge.center_id.unique())

print(len(center))

c_group=full_merge.groupby('center_id')
c_order=list(c_group['num_orders'].agg(np.sum))
print(len(c_order))

plt.figure(figsize=(20,10))
plt.xlabel('Center_ID')
plt.ylabel('No. of orders')
ax=sns.barplot(x=center,y=c_order)
```

```
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())

cuisine=list(full_merge.cuisine.unique())
print(len(cuisine))

cuisine_order=list(full_merge.groupby('cuisine')['num_orders'].agg(np.sum))

print(len(cuisine_order))

plt.figure(figsize=(10,5))
plt.xlabel('Cuisine')
plt.ylabel('No. of orders')
ax=sns.barplot(x=cuisine,y=cuisine_order)
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())

cat=full_merge.category.unique()
cat_order=list(full_merge.groupby('category')['num_orders'].agg(np.sum))

plt.figure(figsize=(20,10))
plt.xlabel('Category')
plt.ylabel('No. of orders')
ax=sns.barplot(x=cat,y=cat_order)
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())

import math
prediction=pd.read_csv('/content/drive/My Drive/Colab Notebooks/results.csv')
prediction.num_orders=prediction.num_orders.astype('int')
full_test=pd.merge(full_test,prediction,how='left',left_on='id',right_on='id')

full_test.head()

lt=list(full_test.meal_id.unique())
print(len(lt))
```

```
meal_groupt=full_test.groupby('meal_id')
order_sortt=list(meal_groupt['num_orders'].agg(np.sum))
print(len(order_sortt))

plt.figure(figsize=(20,10))
plt.xlabel('Meal_ID')
plt.ylabel('No. of orders')
ax=sns.barplot(x=lt,y=order_sortt)
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())

centert=list(full_test.center_id.unique())
c_groupt=full_test.groupby('center_id')
c_ordert=list(c_groupt['num_orders'].agg(np.sum))
print(len(c_ordert))

plt.figure(figsize=(20,10))
plt.xlabel('Center_ID')
plt.ylabel('No. of orders')
ax=sns.barplot(x=centert,y=c_ordert)
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())

cuisinet=list(full_test.cuisine.unique())

cuisine_ordert=list(full_test.groupby('cuisine')['num_orders'].agg(np.sum))
plt.figure(figsize=(10,5))
plt.xlabel('Cuisine')
plt.ylabel('No. of orders')
ax=sns.barplot(x=cuisinet,y=cuisine_ordert)
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())

catt=full_test.category.unique()
cat_ordert=list(full_test.groupby('category')['num_orders'].agg(np.sum))
plt.figure(figsize=(20,10))
plt.xlabel('Category')
plt.ylabel('No. of orders')
ax=sns.barplot(x=catt,y=cat_ordert)
ax.set_xticklabels(ax.get_xticklabels(),rotation=60,horizontalalignment='right')
ax.set_yticklabels(ax.get_yticklabels())
```