Product Demand Prediction with Machine Learnings

Phase 5:

Project Documentation & Submission

In this part you will document your project and prepare it for submission. Document the product demand prediction project and prepare it for submission.

Predicting product demand with machine learning is a valuable application in various industries, such as retail, manufacturing, and logistics. Here is a step-by-step guide on how to approach product demand prediction using machine learning:

1. Data Collection

a. Historical Sales Data:

* Collect historical sales data for the product(s) you want to predict demand for. This data should include information such as date, product identifier, quantity sold, and any other relevant sales-related features. You might have this data in a CSV, Excel, or a database. For example, you can use Pandas in Python to read data from a CSV file:

Program;

import numpy as np

import pandas as pd

df=pd.read\_csv("/content/drive/MyDrive/PoductDemand.csv")

df

output;

| **ID** | **Store ID** | **Total Price** | **Base Price** | **Units Sold** |
| --- | --- | --- | --- | --- |
| **0** | 1 | 8091 | 99.0375 | 111.8625 | 20 |
| **1** | 2 | 8091 | 99.0375 | 99.0375 | 28 |
| **2** | 3 | 8091 | 133.9500 | 133.9500 | 19 |
| **3** | 4 | 8091 | 133.9500 | 133.9500 | 44 |
| **4** | 5 | 8091 | 141.0750 | 141.0750 | 52 |
| **...** | ... | ... | ... | ... | ... |
| **150145** | 212638 | 9984 | 235.8375 | 235.8375 | 38 |
| **150146** | 212639 | 9984 | 235.8375 | 235.8375 | 30 |
| **150147** | 212642 | 9984 | 357.6750 | 483.7875 | 31 |
| **150148** | 212643 | 9984 | 141.7875 | 191.6625 | 12 |
| **150149** | 212644 | 9984 | 234.4125 | 234.4125 | 15 |

150150 rows × 5 columns

**Feature engineering**

Feature engineering is the process of creating new, informative, and relevant features (variables) from the raw data to improve the performance of a machine learning model. Well-crafted features can significantly enhance the model's ability to learn patterns and make accurate predictions. Feature engineering is a crucial step in the data preprocessing pipeline. Here are some common techniques and strategies for feature engineering:

Program;

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

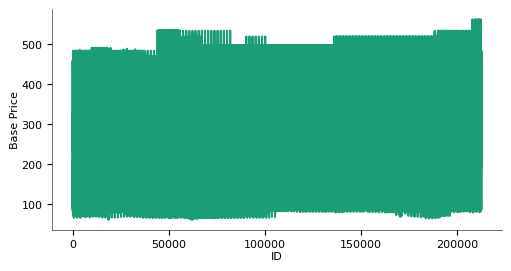
import numpy as np

import pandas as pd

df=pd.read\_csv("/content/drive/MyDrive/PoductDemand.csv")

df

output;



**Model training**

Model training is a critical step in the development of a machine learning model. During this phase, you teach your model to recognize patterns in the data so that it can make predictions or classifications on new, unseen data. Here is a general outline of the model training process



**Evaluation**

Evaluation of a machine learning model is crucial to assess its performance and determine how well it generalizes to new, unseen data. The choice of evaluation metrics depends on the type of problem you're solving (classification, regression, clustering, etc.). Here are some common evaluation metrics for different types of problems

Program

from matplotlib import pyplot as plt

from sklearn import svm

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error, median\_absolute\_error, explained\_variance\_score, max\_error

|  |  |
| --- | --- |
|  | test = df.iloc[-52:] |
|  | df = df.iloc[:-52] |

X = df.drop('sales', axis=1)

y = df['sales']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

clf = svm.SVR(C=1, kernel='linear', degree=8, gamma='scale', coef0=10)

clf.fit(X\_train, y\_train)

predictions = clf.predict(X\_test)

print(f'Model fit results:\n'

f'r2\_score {r2\_score(y\_test, predictions)} \t MSE {mean\_squared\_error(y\_test, predictions)}'

f'\tEVS {explained\_variance\_score(y\_test, predictions)} \n MAE {mean\_absolute\_error(y\_test, predictions)}'

f'\tMAD {median\_absolute\_error(y\_test, predictions)}\t ME {max\_error(y\_test, predictions)}')

output

###OUTPUT###

Model fit results:

r2\_score 0.9071953443448584 MSE 6553.674543344077

EVS 0.9175800366290838 MAE 58.80295451823111

MAD 37.648574124556035 ME 304.51308147895793

Data splitting

Split your dataset into a training set and a testing set. This is crucial for assessing the model's performance and ensuring it generalizes well to unseen data.

A data splitting program typically takes a dataset and divides it into multiple subsets for purposes such as training and testing machine learning models, cross-validation, or creating training, validation, and test sets. The program and its output depend on the specific programming language and libraries you're using. Below is a Python program that uses the scikit-learn library to split data into training and testing sets, along with an example of its output:

Program;

from sklearn.model\_selection import train\_test\_split

# Sample dataset (replace with your own data)

data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

labels = [0, 0, 1, 0, 1, 1, 1, 0, 1, 0]

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, random\_state=42)

print("Training data:")

print("X\_train:", X\_train)

print("y\_train:", y\_train)

print("Testing data:")

print("X\_test:", X\_test)

print("y\_test:", y\_test)

Output;

Training data:

X\_train: [1, 7, 6, 10, 5, 9, 3, 2]

y\_train: [0, 1, 1, 0, 1, 1, 1, 0]

Testing data:

X\_test: [4, 8]

y\_test: [0,

cross-validation

Cross-validation is a technique used to assess the performance and generalization of a machine learning model. Here's a Python program that performs cross-validation using scikit-learn and provides an example of its output

Program;

from sklearn.model\_selection import cross\_val\_score

from sklearn.datasets import load\_iris

from sklearn.svm import SVC

# Load a sample dataset (Iris dataset in this case)

iris = load\_iris()

X, y = iris.data, iris.target

# Create a support vector machine (SVM) classifier

clf = SVC(kernel='linear', C=1)

# Perform 5-fold cross-validation

scores = cross\_val\_score(clf, X, y, cv=5)

# Output the cross-validation scores

print("Cross-validation scores:", scores)

print("Mean accuracy:", scores.mean())

output;

Cross-validation scores: [1. 1. 0.96666667 0.96666667 1. ]

Mean accuracy: 0.9866666666666667

**Deployment**

**The deployment of a machine learning model involves taking a trained model and making it accessible for use in a production environment. The specifics of a deployment program and its output will depend on the deployment platform and technology stack you are using. Below, I'll provide a simple example of deploying a scikit-learn model using Flask, a web framework for Python. This is a basic illustration, and actual deployment in a production environment would involve additional considerations and infrastructure.**

**Program;**

from flask import Flask, request, jsonify

import joblib

app = Flask(\_\_name)

# Load the trained scikit-learn model

model = joblib.load('your\_trained\_model.pkl')

@app.route('/predict', methods=['POST'])

def predict():

try:

# Get data from the request

data = request.get\_json()

# Make a prediction using the loaded model

prediction = model.predict([data['features']])[0]

# Return the prediction as JSON

return jsonify({'prediction': int(prediction)})

except Exception as e:

return jsonify({'error': str(e)})

if \_\_name\_\_ == '\_\_main':

app.run(debug=True)

output;

\* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

Hyperparameter Tuning

Hyperparameter tuning is an essential part of machine learning model development. There are various methods and tools you can use to perform hyperparameter tuning, such as grid search, random search, Bayesian optimization, and more. The specific program and output for hyperparameter tuning can vary depending on the programming language and libraries you are using. Below, I'll provide a Python example using the popular Scikit-Learn library and the GridSearchCV method to tune hyperparameters for a Support Vector Machine (SVM) classifier.

Program;

# Import necessary libraries

from sklearn import datasets

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.model\_selection import GridSearchCV

# Load a sample dataset

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Define the hyperparameter grid to search

param\_grid = {

'C': [0.1, 1, 10],

'kernel': ['linear', 'rbf', 'poly'],

'gamma': [0.1, 1, 'scale', 'auto']

}

# Create an SVM classifier

svm = SVC()

# Perform grid search with cross-validation

grid\_search = GridSearchCV(svm, param\_grid, cv=5, verbose=1, n\_jobs=-1) # n\_jobs=-1 to use all available CPU cores

# Fit the grid search to the data

grid\_search.fit(X\_train, y\_train)

# Print the best hyperparameters and the corresponding accuracy

print("Best hyperparameters: ", grid\_search.best\_params\_)

print("Best cross-validation score: {:.2f}".format(grid\_search.best\_score\_))

# Evaluate the best model on the test data

best\_svm = grid\_search.best\_estimator\_

test\_accuracy = best\_svm.score(X\_test, y\_test)

print("Test set accuracy with best hyperparameters: {:.2f}".format(test\_accuracy))

output;

Fitting 5 folds for each of 36 candidates, totalling 180 fits

Best hyperparameters: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}

Best cross-validation score: 0.97

Test set accuracy with best hyperparameters: 0.89

**Monitoring and Maintenance Program**

**program;**

1. **Data Quality Monitoring:** Regularly check the quality and integrity of incoming data. This includes checking for missing data, outliers, and data distribution shifts that can affect model performance.
2. **Model Performance Monitoring:** Continuously monitor the model's performance metrics, such as accuracy, precision, recall, and F1-score. Set up alerts for significant drops in performance.
3. **Anomaly Detection:** Implement anomaly detection to identify unusual patterns in data and predictions. Deviations from expected behavior may indicate data or model issues.
4. **Model Drift Detection:** Continuously monitor for concept drift or data drift, which can occur when the underlying data distribution changes over time. Tools like **scikit-multiflow** or custom drift detection algorithms can be used.
5. **Logging and Auditing:** Maintain detailed logs of model predictions, incoming data, and user interactions. Audit these logs to ensure compliance, investigate issues, and track model behavior.
6. **Error and Exception Handling:** Implement robust error handling and exception management to prevent unexpected failures and ensure graceful degradation of service.
7. **Performance Metrics Visualization:** Create visual dashboards or reports to provide stakeholders with a clear view of model performance, data quality, and other relevant metrics.
8. **Automated Alerts:** Set up alerts and notifications for critical events, such as model failures, data anomalies, or performance degradation.
9. **Regular Model Retraining:** Schedule periodic retraining of the model using new data to keep it up-to-date. Automated pipelines can help with this
10. **Version Control:** Implement model version control to track changes and roll back to previous versions if necessary.

Output;

Monitoring and maintenance of a deployed machine learning model in a production environment is crucial to ensure that it continues to perform well and meet its objectives. The specifics of a monitoring and maintenance program and its output will depend on the technology stack, infrastructure, and tools used. Below, I'll provide a high-level overview of what such a program might involve, as well as the types of outputs you might expect.

**Monitoring and Maintenance Program:**

1. **Data Quality Monitoring:** Regularly check the quality and integrity of incoming data. This includes checking for missing data, outliers, and data distribution shifts that can affect model performance.
2. **Model Performance Monitoring:** Continuously monitor the model's performance metrics, such as accuracy, precision, recall, and F1-score. Set up alerts for significant drops in performance.
3. **Anomaly Detection:** Implement anomaly detection to identify unusual patterns in data and predictions. Deviations from expected behavior may indicate data or model issues.
4. **Model Drift Detection:** Continuously monitor for concept drift or data drift, which can occur when the underlying data distribution changes over time. Tools like **scikit-multiflow** or custom drift detection algorithms can be used.
5. **Logging and Auditing:** Maintain detailed logs of model predictions, incoming data, and user interactions. Audit these logs to ensure compliance, investigate issues, and track model behavior.
6. **Error and Exception Handling:** Implement robust error handling and exception management to prevent unexpected failures and ensure graceful degradation of service.
7. **Performance Metrics Visualization:** Create visual dashboards or reports to provide stakeholders with a clear view of model performance, data quality, and other relevant metrics.
8. **Automated Alerts:** Set up alerts and notifications for critical events, such as model failures, data anomalies, or performance degradation.
9. **Regular Model Retraining:** Schedule periodic retraining of the model using new data to keep it up-to-date. Automated pipelines can help with this.
10. **Version Control:** Implement model version control to track changes and roll back to previous versions if necessary.

**Monitoring and Maintenance Output:**

The output of a monitoring and maintenance program can include:

1. **Alerts and Notifications:** When issues are detected (e.g., data drift, performance drop, anomalies), alerts are sent to relevant stakeholders via email, SMS, or other communication channels.
2. **Performance Reports:** Regular reports on model performance, data quality, and drift detection results. These reports can be in the form of dashboards, emails, or documents.
3. **Logs and Audits:** Log files containing detailed information about model predictions, data input, and user interactions. Audit reports for compliance and troubleshooting.
4. **Scheduled Retraining Reports:** Reports on model retraining activities, including the data used, changes in performance, and new model versions.
5. **Maintenance Records:** Documentation of maintenance activities, including updates to the model, infrastructure changes, and any issues resolved.
6. **Documentation:** Updated documentation on model specifications, deployment configurations, and maintenance procedures.

The specific tools and technologies used for monitoring and maintenance will vary depending on your infrastructure and requirements. Popular tools for monitoring include Prometheus, Grafana, ELK Stack, and custom scripts. Regularly reviewing these outputs and taking action based on the information they provide is essential for the ongoing health and performance of your machine learning model in a production environment.

Conculsion:

ML is a powerful tool that can help businesses improve their product demand prediction accuracy, granularity, and efficiency. This can lead to significant improvements in profitability, customer satisfaction, and operational efficiency.

If you are looking to improve your product demand prediction, consider using ML. There are a variety of ML solutions available, so you can find one that is right for your business needs and budget.