**FUTURE SALES PREDICTION USING MACHINE LEARNING**

**PHASE 4 SUBMISSION PROJECT**

**PROJECT TITLE:** **Future sales prediction**

**TOPIC:** **Building the project by performing feature engineering, model training and evaluation.**

**Introduction:**

* Sales prediction — also commonly called **sales forecasting** — is the process of estimating future sales by predicting the amount of product or services an individual salesperson, a sales team, or a company is likely to sell in a fixed time period i.e. next week, month, quarter, or year.
* Feature selection is the process of identifying and selecting the most relevant features from a dataset to improve the performance of a machine learning model. This is an important step in building SALES PREDICTION model, as it can help to reduce over fitting and improve the generalization ability of the model.
* Model training is the process of feeding the selected features to a machine learning algorithm and allowing it to learn the relationship between the features and the target variable .Once the model is trained, it can be used to predict future sales as by given their features.

**Overview of the process:**

The following is an overview of the process of future sales prediction model by feature selection, model training, and evaluation:

**1. Prepare the data:** This includes cleaning the data, removing outliers, and handling missing values.

**2. Perform feature selection:** This can be done using a variety of methods, such as correlation analysis, information gain, and recursive feature elimination.

**3. Train the model:** There are many different machine learning algorithms that can be used for future sales prediction. Some popular choices include linear regression, random forests, and gradient boosting machines.

**4. Evaluate the model:** This can be done by calculating the mean squared error (MSE) or the root mean squared error (RMSE) of the model's predictions on the held-out test set.

**5. Deploy the model:** Once the model has been evaluated and found to be performing well, it can be deployed to production so that it can be used to predict the house prices of new houses.

**Data fields description:**

ID - an Id that represents a (Shop, Item) tuple within the test set

shop\_id - unique identifier of a shop

item\_id - unique identifier of a product

item\_category\_id - unique identifier of item category

date\_block\_num - a consecutive month number, used for convenience. January 2013 is 0, February 2013 is 1,..., October 2015 is 33

date - date in format dd/mm/yyyy

item\_cnt\_day - number of products sold. You are predicting a monthly amount of this measure

item\_price - current price of an item

item\_name - name of item

shop\_name - name of shop

item\_category\_name - name of item category

**Data preprocessing:**

train\_monthly = lk\_train[['date', 'date\_block\_num', 'shop\_id', 'item\_category\_id', 'item\_id', 'item\_price', 'item\_cnt\_day']]

train\_monthly = train\_monthly.sort\_values('date').groupby(['date\_block\_num', 'shop\_id', 'item\_category\_id', 'item\_id'], as\_index=False)

train\_monthly = train\_monthly.agg({'item\_price':['sum', 'mean'], 'item\_cnt\_day':['sum', 'mean','count']})

**Feature engineering:**

Feature engineering is a crucial aspect of building a house price prediction model using machine learning. It involves creating new features, transforming existing ones, and selecting the most relevant variables to improve the model's predictive power.

Unitary item prices.

train\_monthly['item\_price\_unit'] = train\_monthly['item\_price'] // train\_monthly['item\_cnt']

train\_monthly['item\_price\_unit'].fillna(0, inplace=True)

Group based features.

gp\_item\_price = train\_monthly.sort\_values('date\_block\_num').groupby(['item\_id'], as\_index=False).agg({'item\_price':[np.min, np.max]})

gp\_item\_price.columns = ['item\_id', 'hist\_min\_item\_price', 'hist\_max\_item\_price']

train\_monthly = pd.merge(train\_monthly, gp\_item\_price, on='item\_id', how='left')

# Min value

f\_min = lambda x: x.rolling(window=3, min\_periods=1).min()

# Max value

f\_max = lambda x: x.rolling(window=3, min\_periods=1).max()

# Mean value

f\_mean = lambda x: x.rolling(window=3, min\_periods=1).mean()

# Standard deviation

f\_std = lambda x: x.rolling(window=3, min\_periods=1).std()

function\_list = [f\_min, f\_max, f\_mean, f\_std]

function\_name = ['min', 'max', 'mean', 'std']

for i in range(len(function\_list)):

train\_monthly[('item\_cnt\_%s' % function\_name[i])] = train\_monthly.sort\_values('date\_block\_num').groupby(['shop\_id', 'item\_category\_id', 'item\_id'])['item\_cnt'].apply(function\_list[i])

# Fill the empty std features with 0

train\_monthly['item\_cnt\_std'].fillna(0, inplace=True)

Lag based features.

lag\_list = [1, 2, 3]

**Build test set:**

latest\_records = pd.concat([train\_set, validation\_set]).drop\_duplicates(subset=['shop\_id', 'item\_id'], keep='last')

X\_test = pd.merge(test, latest\_records, on=['shop\_id', 'item\_id'], how='left', suffixes=['', '\_'])

X\_test['year'] = 2015

X\_test['month'] = 9

X\_test.drop('item\_cnt\_month', axis=1, inplace=True)

X\_test[int\_features] = X\_test[int\_features].astype('int32')

X\_test = X\_test[X\_train.columns]

Replacing missing values.

sets = [X\_train, X\_validation, X\_test]

**Modeling the data:**

Choose a machine learning algorithm. There are a number of different machine learning algorithms that can be used for house price prediction, such as linear regression, ridge regression, lasso regression, decision trees, and random forests.

For example, you could use the model to predict the price of a house that you are interested in buying.

1. Prepare the data. This involves cleaning the data, removing any errors or inconsistencies, and transforming the data into a format that is compatible with the machine learning algorithm that you will be using.

2. Split the data into training and test sets. The training set will be used to train the model, and the test set will be used to evaluate the performance of the model on unseen data.

3. Choose a machine learning algorithm. There are a number of different machine learning algorithms that can be used for future sales prediction, such as linear regression, ridge regression, lasso regression, decision trees, and random forests.

4. Tune the hyperparameters of the algorithm. The hyperparameters of a machine learning algorithm are parameters that control the learning process. It is important to tune the hyperparameters of the algorithm to optimize its performance.

5. Train the model on the training set. This involves feeding the training data to the model and allowing it to learn the relationships between the features and sales prices.

6. Evaluate the model on the test set. This involves feeding the test data to the model and measuring how well it predicts the sales prices.

**Tree based models:**

Catboost

cat\_features = [0, 1, 7, 8]

catboost\_model = CatBoostRegressor(

iterations=500,

max\_ctr\_complexity=4,

random\_seed=0,

od\_type='Iter',

od\_wait=25,

verbose=50,

depth=4

)

catboost\_model.fit(

X\_train, Y\_train,

cat\_features=cat\_features,

eval\_set=(X\_validation, Y\_validation)

)

**XGBoost:**

# Use only part of features on XGBoost.

xgb\_features = ['item\_cnt','item\_cnt\_mean', 'item\_cnt\_std', 'item\_cnt\_shifted1',

'item\_cnt\_shifted2', 'item\_cnt\_shifted3', 'shop\_mean',

'shop\_item\_mean', 'item\_trend', 'mean\_item\_cnt']

xgb\_train = X\_train[xgb\_features]

xgb\_val = X\_validation[xgb\_features]

xgb\_test = X\_test[xgb\_features]

xgb\_model = XGBRegressor(max\_depth=8,

n\_estimators=500,

min\_child\_weight=1000,

colsample\_bytree=0.7,

subsample=0.7,

eta=0.3,

seed=0)

xgb\_model.fit(xgb\_train,

Y\_train,

eval\_metric="rmse",

eval\_set=[(xgb\_train, Y\_train), (xgb\_val, Y\_validation)],

verbose=20,

early\_stopping\_rounds=20)

[20:29:47] Tree method is automatically selected to be 'approx' for faster speed. To use old behavior (exact greedy algorithm on single machine), set tree\_method to 'exact'.

[0] validation\_0-rmse:0.937872 validation\_1-rmse:0.924623

**Random forest:**

# Use only part of features on random forest.

rf\_features = ['shop\_id', 'item\_id', 'item\_cnt', 'transactions', 'year', 'item\_cnt\_mean', 'item\_cnt\_std', 'item\_cnt\_shifted1', 'shop\_mean', 'item\_mean', 'item\_trend', 'mean\_item\_cnt']

rf\_train = X\_train[rf\_features]

rf\_val = X\_validation[rf\_features]

rf\_test = X\_test[rf\_features]

rf\_model = RandomForestRegressor(n\_estimators=50, max\_depth=7, random\_state=0, n\_jobs=-1)

rf\_model.fit(rf\_train, Y\_train)

RandomForestRegressor(bootstrap=True, criterion='mse', max\_depth=7,

max\_features='auto', max\_leaf\_nodes=None,

min\_impurity\_decrease=0.0, min\_impurity\_split=None,

min\_samples\_leaf=1, min\_samples\_split=2,

min\_weight\_fraction\_leaf=0.0, n\_estimators=50, n\_jobs=-1,

oob\_score=False, random\_state=0, verbose=0, warm\_start=False)

rf\_train\_pred = rf\_model.predict(rf\_train)

rf\_val\_pred = rf\_model.predict(rf\_val)

rf\_test\_pred = rf\_model.predict(rf\_test)

Train rmse: 0.6985868322226099

Validation rmse: 0.776123635046122

**Linear models**

**Linear Regression:**

# Use only part of features on linear Regression.

lr\_features = ['item\_cnt', 'item\_cnt\_shifted1', 'item\_trend', 'mean\_item\_cnt', 'shop\_mean']

lr\_train = X\_train[lr\_features]

lr\_val = X\_validation[lr\_features]

lr\_test = X\_test[lr\_features]

**Normalizing features:**

lr\_scaler = MinMaxScaler()

lr\_scaler.fit(lr\_train)

lr\_train = lr\_scaler.transform(lr\_train)

lr\_val = lr\_scaler.transform(lr\_val)

lr\_test = lr\_scaler.transform(lr\_test)

lr\_model = LinearRegression(n\_jobs=-1)

lr\_model.fit(lr\_train, Y\_train)

LinearRegression(copy\_X=True, fit\_intercept=True, n\_jobs=-1, normalize=False)

lr\_train\_pred = lr\_model.predict(lr\_train)

lr\_val\_pred = lr\_model.predict(lr\_val)

lr\_test\_pred = lr\_model.predict(lr\_test)

Train rmse: 0.7347132326333327

Validation rmse: 0.7755311093532269

**Clustering models:**

KNN Regressor

# Use only part of features on KNN.

knn\_features = ['item\_cnt', 'item\_cnt\_mean', 'item\_cnt\_std', 'item\_cnt\_shifted1',

'item\_cnt\_shifted2', 'shop\_mean', 'shop\_item\_mean',

'item\_trend', 'mean\_item\_cnt']

X\_train\_sampled = X\_train[:100000]

Y\_train\_sampled = Y\_train[:100000]

knn\_train = X\_train\_sampled[knn\_features]

knn\_val = X\_validation[knn\_features]

knn\_test = X\_test[knn\_features]

**Normalizing features:**

knn\_scaler = MinMaxScaler()

knn\_scaler.fit(knn\_train)

knn\_train = knn\_scaler.transform(knn\_train)

knn\_val = knn\_scaler.transform(knn\_val)

knn\_test = knn\_scaler.transform(knn\_test)

knn\_model = KNeighborsRegressor(n\_neighbors=9, leaf\_size=13, n\_jobs=-1)

knn\_model.fit(knn\_train, Y\_train\_sampled)

KNeighborsRegressor(algorithm='auto', leaf\_size=13, metric='minkowski',

metric\_params=None, n\_jobs=-1, n\_neighbors=9, p=2,

weights='uniform')

knn\_train\_pred = knn\_model.predict(knn\_train)

knn\_val\_pred = knn\_model.predict(knn\_val)

knn\_test\_pred = knn\_model.predict(knn\_test)

Train rmse: 0.48661440612348666

Validation rmse: 0.80036105644979

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | catboost | xgbm | random\_forest | linear\_regression | knn | label |
| 0 | 0.74 | 0.70 | 0.62 | 0.60 | 1.00 | 0 |
| 1 | 1.03 | 0.64 | 0.56 | 0.04 | 0.78 | 0 |
| 2 | 0.03 | 0.10 | 0.21 | 0.04 | 0.00 | 0 |
| 3 | 1.31 | 0.81 | 0.95 | 0.04 | 0.11 | 4 |
| 4 | 1.63 | 1.59 | 1.57 | 1.46 | 0.67 | 1 |
| 5 | 0.89 | 0.64 | 0.46 | 0.04 | 0.11 | 1 |
| 6 | 0.54 | 0.43 | 0.21 | 0.04 | 0.11 | 0 |
| 7 | 0.03 | 0.10 | 0.06 | 0.04 | 0.00 | 1 |
| 8 | 0.52 | 0.38 | 0.21 | 0.04 | 0.11 | 0 |
| 9 | 1.89 | 1.73 | 1.13 | 1.15 | 2.89 | 2 |
| 10 | 0.83 | 0.53 | 0.27 | 0.04 | 0.56 | 0 |
| 11 | 0.63 | 0.43 | 0.41 | 0.04 | 0.11 | 0 |
| 12 | 0.96 | 0.64 | 0.67 | 0.04 | 0.11 | 0 |
| 13 | 0.83 | 0.53 | 0.46 | 0.04 | 0.33 | 0 |
| 14 | 0.07 | 0.13 | 0.06 | 0.04 | 0.00 | 0 |
| 15 | 0.65 | 0.46 | 0.28 | 0.04 | 0.22 | 1 |
| 16 | 0.31 | 0.28 | 0.20 | 0.04 | 0.11 | 0 |
| 17 | 0.68 | 0.53 | 0.28 | 0.04 | 0.11 | 0 |
| 18 | 0.59 | 0.57 | 0.57 | 0.60 | 0.89 | 0 |
| 19 | 0.48 | 0.52 | 0.60 | 1.08 | 0.33 | 0 |

**Output dataframe:**

prediction\_df = pd.DataFrame(test['ID'], columns=['ID'])

prediction\_df['item\_cnt\_month'] = final\_predictions.clip(0., 20.)

prediction\_df.to\_csv('submission.csv', index=False)

prediction\_df.head(10)

|  |  |  |
| --- | --- | --- |
| s.no | ID | item\_cnt\_month |
| 0 | 0 | 0.82 |
| 1 | 1 | 0.08 |
| 2 | 2 | 1.26 |
| 3 | 3 | 0.06 |
| 4 | 4 | 0.08 |
| 5 | 5 | 0.94 |
| 6 | 6 | 1.23 |
| 7 | 7 | 0.21 |
| 8 | 8 | 1.98 |
| 9 | 9 | 0.06 |

**Model evaluation:**

* Model evaluation is the process of assessing the performance of a machine learning model on unseen data. This is important to ensure that the model will generalize well to new data.
* There are a number of different metrics that can be used to evaluate the performance of future sales prediction model. Some of the most common metrics include:
* **Mean squared error (MSE):** This metric measures the average squared difference between the predicted and actual sales prices.
* **Root mean squared error (RMSE):** This metric is the square root of the MSE.
* **Mean absolute error (MAE):** This metric measures the average absolute difference between the predicted and actual sales prices
* **R-squared:** This metric measures how well the model explains the variation in the actual sales prices.

**In addition to these metrics, it is also important to consider the following factors when evaluating a sales price prediction model:**

* **Bias**: Bias is the tendency of a model to consistently over- or underestimate sales prices.
* **Variance:** Variance is the measure of how much the predictions of a model vary around the true sales prices.
* **Interpretability:** Interpretability is the ability to understand how the model makes its predictions.

**Evaluation of Predicted Data:**

In [18]:

plt.figure(figsize=(12,6))

plt.plot(np.arange(len(Y\_test)), Y\_test, label='Actual Trend')

plt.plot(np.arange(len(Y\_test)), Prediction5, label='Predicted Trend')

plt.xlabel('Data')

plt.ylabel('Trend')

plt.legend()

plt.title('Actual vs Predicted')

Out[18]:

Text(0.5, 1.0, 'Actual vs Predicted')

**Conclusion:**

In the quest to build an accurate and reliable future sales prediction model, we have embarked on a journey that encompasses critical phases, from feature selection to model training and evaluation. Each of these stages plays an indispensable role in crafting a model that can provide meaningful insights and estimates for one of the most significant financial decisions individuals and businesses make—real estate transactions.

* Model training is where the model's predictive power is forged. We have explored a variety of regression techniques, fine-tuning their parameters to learn from historical data patterns. This step allows the model to capture the intricate relationships between features and sales prices, giving it the ability to generalize beyond the training dataset.
* Finally, model evaluation is the litmus test for our predictive prowess. Using metrics like Mean Squared Error, Root Mean Squared Error, Mean Absolute Error, and R-squared, we've quantified the model's performance. This phase provides us with the confidence to trust the model's predictions and assess its ability to adapt to unseen data.

In the ever-evolving world of real estate and finance, a robust sales price prediction model is an invaluable tool. It aids buyers, sellers, and investors in making informed decisions, mitigating risks, and seizing opportunities. As more data becomes available and market dynamics change, the model can be retrained and refined to maintain its accuracy.