POLIMI GRADUATE SCHOOL OF MANAGEMENT

03/04/2024

MACHINE LEARNING & BUSINESS APPLICATIONS

Brief Report

Sahitya Chaudhary
Andrea Sereni
Raphaël Rortais

SUMMARY

A) Problem Statement :

Yojo.com, a prominent player in the online shopping industry, seeks to enhance its customer satisfaction by proactively addressing potential issues highlighted in product reviews. Understanding that popular reviews significantly influence customer satisfaction, Yojo.com aims to predict the popularity of reviews based on various textual features. **This predictive capability will enable the company to identify and intervene in case of potential negative feedback that could garner significant attention.**

B) Problematic :

Can Yojo.com predict the popularity of product reviews based on textual features, allowing them to proactively address potential issues and enhance customer satisfaction in the competitive online shopping market?

C) Motivations and Approach :

Our motivations for undertaking this project are to enhance customer satisfaction and **improve business performance** for Yojo.com in the competitive online shopping market. By leveraging machine learning techniques to predict the popularity of product reviews based on textual features, we aim to proactively address potential issues highlighted in these reviews. This proactive approach will enable Yojo.com to maintain **high levels of customer satisfaction, mitigate negative impacts on their reputation, and ultimately drive growth and success in the online shopping industry.**

A) The first observations

```
12  product_category                28000 non-null  object
25  day                             28000 non-null  object
```

Upon reviewing the model.csv dataset, it was observed that two columns contain categorical values. These categorical values, representing the day of publication and product category, **need to be converted into numerical format for inclusion in the model**. To address this, the days column will be mapped to numerical values, and the product category column will be encoded using dummies. These pre-processing steps will ensure that the dataset does not raise an error while **training.**

```
33  global_rate_positive_words      28000 non-null  float64
34  global_rate_negative_words      28000 non-null  float64
```

After reviewing the dataset, our focus shifts to identifying redundant columns. We'll run a function to analyze the covariance between columns and generate a list of potential candidates for removal. This step aims to streamline the dataset by eliminating unnecessary features, such as those with redundant information like the percentage of positive and negative votes.

III- Modelling Process

This **covariance analysis helped confirming the redundant columns.** Specifically, we checked for columns with a covariance of **50% or higher** with any other column. Any columns meeting this criterion were considered redundant and **flagged for removal**.
Upon review, we found that the identified list of columns could be explained by others in our analysis. Therefore, we proceeded to **remove these redundant columns from the dataset**. This step ensured that our dataset remained concise and contained only relevant features, **optimizing** it for further analysis and modeling.

```
# Calculate the correlation matrix
corr_matrix = df_train_v2.corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find features with correlation greater than a specific threshold, in our case, 50%
threshold = 0.5
to_drop = [column for column in upper.columns if any(upper[column] > threshold)]
```

We utilized this code to calculate the correlation matrix of the preprocessed dataset. Subsequently, we identified columns with high covariance, specifically those with a covariance **greater than 50%** with another column. These columns were marked for removal as they were deemed redundant.

```
[ ]  df_train_v2 = df_train_v2.drop(to_drop, axis=1)
     df_train_v2.head()
```

Following the identification of redundant columns, we dropped these columns from the dataset using the drop() function along the specified axis (axis=1 denotes columns). The resulting dataset was then displayed to inspect the changes.

III- Modelling Process

POLIMI GRADUATE SCHOOL OF MANAGEMENT

Outlier Detection :

In this process, we begin by dividing our dataset into quartiles to understand where most of the data lies, defining the first quartile (Q1) as the bottom 25% and the third quartile (Q3) as the top 25%. Utilizing these quartiles, we calculate the **Interquartile Range** (IQR), which signifies the **spread** of the "middle" numbers.

We then employ the **IQR to identify and CAP the present outliers**, which are numbers significantly distant from the rest, using a **criterion of 1.5 times the IQR above Q3 or below Q1**.
We believe that those outliers are TRUE and not errors, we decided therefore to cap instead of dropping them. This means that each row with a " shares" value above our upper_bound (and vice-versa) will have its " shares" value assigned a fixed threshold. This helps us reducing the skewness and to focus the training on the majority of the dataset. Finally, we assess the cleaned dataset's summary statistics to confirm that the removal of outliers has not adversely affected the dataset's representativeness or integrity, thereby enhancing the reliability of our subsequent analysis.

```
#Cap the outiliers
df_train_v2[' shares'] = df_train_v2[' shares'].apply(lambda x: max(min(x, upper_bound), lower_bound))
df_train_v2.describe()
```

Before

```
df_train_v1[" shares"].describe()
✓ 0.0s
```

```
count     28000.000000
mean       3408.232750
std       12578.941208
min           5.000000
25%         942.000000
50%        1400.000000
75%        2700.000000
max      843300.000000
Name:  shares, dtype: float64
```

After

```
df_train_v2[' shares'].describe()
✓ 0.0s
```

```
count    28000.000000
mean      2083.280857
std       1556.179400
min          5.000000
25%        942.000000
50%       1400.000000
75%       2700.000000
max       5337.000000
Name:  shares, dtype: float64
```

→ Separating the training set from the test set :

In this process, we **split our dataset into two parts**: features (X) and target variable (y). First, we create the feature dataset (X) by dropping the column named " shares" from the original dataset. Next, we isolate the target variable (y) by selecting only the " shares" column. Following this separation, we further **divide our data into training and testing sets using the train_test_split function from the scikit-learn library.**

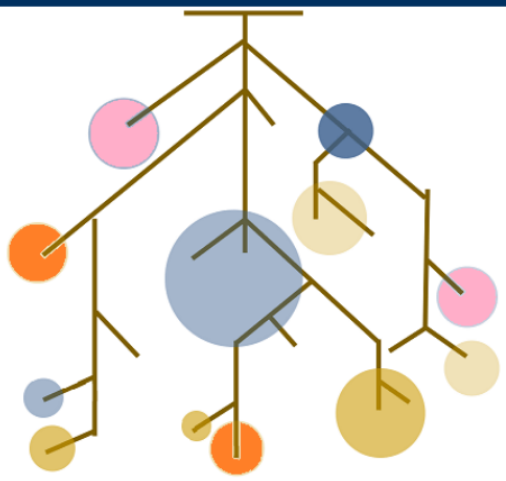
The test_size parameter specifies the proportion of the dataset to include in the test set, with 0.2 indicating that **20% of the data will be reserved for testing**. Additionally, we set the random_state parameter to 40 to ensure reproducibility of the split. This division allows us to train our machine learning model on a subset of the data **(training set)** and evaluate its performance on unseen data **(test set)**, facilitating reliable model assessment and validation.

```
X = df_train_v2.drop(columns=" shares")
y = df_train_v2[" shares"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In conclusion, our data preprocessing involved identifying and removing outliers to ensure the integrity of our analysis, while separating the dataset into training and testing sets established a robust framework for model training and evaluation. Reflection on parameters like test_size and random_state in train_test_split is essential for optimizing model performance. Additionally, ongoing validation of our model's performance on unseen data is crucial for real-world reliability.

III- Modelling Process

POLIMI GRADUATE SCHOOL OF MANAGEMENT



Model Fitting :

In this process, we aimed to develop a **decision tree model to predict the number of shares for online articles based on various features**. A decision tree is chosen as it's a simple yet effective method for understanding the relationship between variables and making predictions. This is also useful to create a benchmark for future, more complex models.

We started by creating the decision tree model using scikit-learn. The model was trained using the training data (X_train and y_train), where X_train represents the features and y_train represents the target variable (number of shares).

Once the model was trained, we used it to make predictions on the testing data (X_test). The predicted values (y_pred) were compared with the actual values (y_test) to evaluate the model's performance.

```
[ ]  # Create a linear regression model
     model = DecisionTreeRegressor()

     model.fit(X_train, y_train)

     y_pred_slr_2 = model.predict(X_test)

     mae = metrics.mean_absolute_error(y_test, y_pred_slr_2)


     print(f"The MAE is : {mae}")
```

To assess the model's accuracy, we calculated the mean absolute error **(MAE)** which represents the average absolute difference between the predicted and actual values.

Before Cleaning

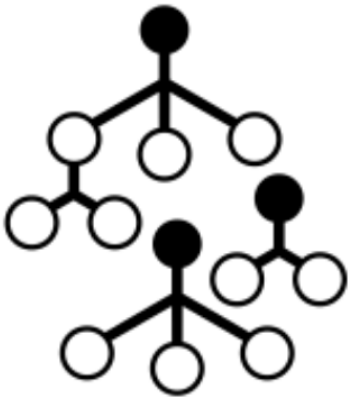
```
The MAE is : 4316.1476785714285
```

After Cleaning

```
The MAE is : 1493.88875
```

Random Forest :

Random Forest offers companies a powerful tool for accurate predictions, particularly in complex datasets with diverse characteristics. Its robustness in handling diverse data types, its ability to capture non-linear relationships, such as ours, and its capacity to provide information on feature importance make it an invaluable tool.



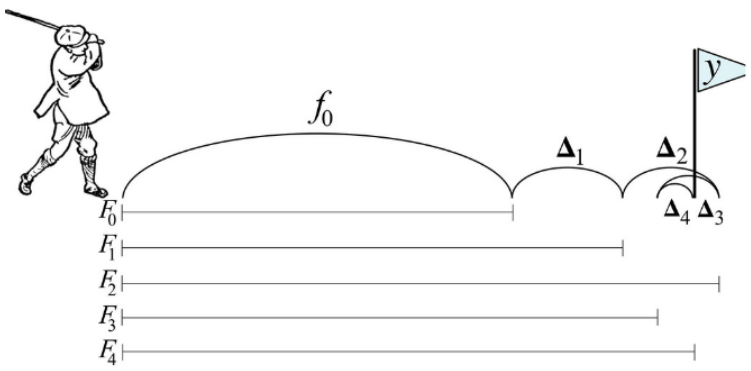
However, its computational complexity, lack of interpretability and the need for careful hyperparameter tuning are problematic. Despite these considerations, if we take its advantages into account, **Random Forest remains a very effective choice for companies looking for accurate and reliable predictive models, which is why we have decided to implement this technique seen in course in our analysis.**

```
[ ]  rf_model = RandomForestRegressor(n_estimators=100, random_state=40)
     rf_model.fit(X_train, y_train)
     y_pred_rf = rf_model.predict(X_test)

     mae = metrics.mean_absolute_error(y_test, y_pred_rf)
     r2 = metrics.r2_score(y_test, y_pred_rf)

     print(f"The MAE is : {mae}")
     print(f"The R2 is : {r2}")
```

```
The MAE is : 1160.8814410714285
The R2 is : 0.1330610929664103
```

In this process, our goal was to predict the number of shares for online articles using a Random Forest regression model. We trained the model with the training data and evaluated its performance using the testing data. By calculating the mean absolute error (MAE) and coefficient of determination (R^2), we assessed the model's accuracy.

III- Modelling Process



Gradient boosting machines :

"is an ensemble machine learning technique that combines the predictions from several models to improve the overall predictive accuracy. It is particularly useful for regression and classification problems." https://deepai.org/

```python
xgb_model = xgb.XGBRegressor(objective ='reg:squarederror', n_estimators=50, learning_rate=0.1, random_state=40)

# Train the model
xgb_model.fit(X_train_v3, y_train_v3)

# Make predictions
y_pred_xb_v3 = xgb_model.predict(X_test_v3)

mae = metrics.mean_absolute_error(y_test_v3, y_pred_xb_v3)
r2 = metrics.r2_score(y_test_v3, y_pred_xb_v3)


print(f"The MAE is : {mae}")
print(f"The R2 is : {r2}")
```

```
The MAE is : 3125.2504384395056
The R2 is : -0.1745354185900283
```

This code sets up and trains an XGBoost regression model to predict the number of shares for online articles. It uses specific parameters like the **objective function for regression, the number of trees (50), and the learning rate (0.1)**. After training, the model makes predictions on the test data and calculates metrics like mean absolute error (MAE) and coefficient of determination (R^2) to evaluate its performance.

```python
# Initialize XGBRegressor
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror', random_state=42)

# Parameter grid for RandomizedSearch
param_grid = {
    "n_estimators": [100, 200, 300, 400, 500],
    "learning_rate": [0.01, 0.05, 0.1, 0.2],
    "max_depth": [2, 3, 4, 5, 6, 7],
    "colsample_bytree": [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],
    "subsample": [0.2, 0.4, 0.6, 0.8, 1.0],
    "gamma": [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
}

# Setup RandomizedSearchCV
random_search = RandomizedSearchCV(estimator=xg_reg, param_distributions=param_grid, n_iter=25,
                                   scoring='neg_mean_absolute_error', cv=5, verbose=2, random_state=42, n_jobs=-1)

# Fit RandomizedSearchCV
random_search.fit(X_train, y_train)

# Best parameters and score
print("Best parameters found: ", random_search.best_params_)
print("Best score found: ", random_search.best_score_)
```

This code sets up a RandomizedSearchCV to find the best hyperparameters for an XGBoost regression model. It defines a parameter grid containing various hyperparameters, such as the number of estimators, learning rate, maximum tree depth, etc

RandomizedSearchCV searches through this parameter grid using random combinations and cross-validation to minimize the negative mean absolute error.

POLIMI GRADUATE SCHOOL OF MANAGEMENT

03/04/2024

MACHINE LEARNING & BUSINESS APPLICATIONS

Thank you for your reading

Sahitya Chaudhary
Andrea Sereni
Raphaël Rortais