Detecting user interface preference using machine learning models

Machine Learning Programming

User UI Preference Classification

By Farbod Fooladi at Shahid-Beheshti-University (SBU)

Abstract

The user interface is the graphical design of a program. This includes the buttons users click, the text they read, images, sliders, text input fields, and anything else the user interacts with. This includes page layouts, transitions, interface animations, and any micro-interactions. Any kind of visual element, interaction, or animation should all be designed. An e-commerce company recently changed their website's user interface, but they don't know how to evaluate their new user interface. Do customers like it? Do they prefer the new interface or the old one?

Contents

Data review:
Our most recent and tested data is provided with the following information:
Train_df:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10605 entries, 0 to 10604
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              10605 non-null  object
 1   Age                     9876 non-null   float64
 2   Gender                  10605 non-null  object
 3   City                    10605 non-null  object
 4   State                   10605 non-null  object
 5   No_of_orders_placed     10070 non-null  float64
 6   Sign_up_date            10605 non-null  object
 7   Last_order_placed_date  10605 non-null  object
 8   is_premium_member       10605 non-null  int64
 9   Women’s_Clothing        10605 non-null  float64
 10  Men’s_Clothing          10605 non-null  float64
 11  Kid’s_Clothing          9957 non-null   float64
 12  Home_&_Living           10010 non-null  float64
 13  Beauty                  10605 non-null  float64
 14  Electronics             10605 non-null  float64
 15  Preferred_Theme         10605 non-null  object
dtypes: float64(8), int64(1), object(7)
memory usage: 1.4+ MB
```

Test_df:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4545 entries, 0 to 4544
Data columns (total 16 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              4545 non-null   object
 1   Age                     4271 non-null   float64
 2   Gender                  4545 non-null   object
 3   City                    4545 non-null   object
 4   State                   4545 non-null   object
 5   No_of_orders_placed     4307 non-null   float64
 6   Sign_up_date            4545 non-null   object
 7   Last_order_placed_date  4545 non-null   object
 8   is_premium_member       4545 non-null   int64
 9   Women’s_Clothing        4545 non-null   float64
 10  Men’s_Clothing          4545 non-null   float64
 11  Kid’s_Clothing          4258 non-null   float64
 12  Home_&_Living           4292 non-null   float64
 13  Beauty                  4545 non-null   float64
 14  Electronics             4545 non-null   float64
 15  Preferred_Theme         4545 non-null   object
dtypes: float64(8), int64(1), object(7)
memory usage: 603.6+ KB
```

As you can see, we have a missing data problem for the following features:

```
Age
No_of_orders_placed
Kid’s_Clothing
Home_&_Living
```

2- Fixing the problem of missing values

We replace missing data with median values:

```
train_df['Age']=train_df['Age'].fillna(train_df['Age'].median())
test_df['Age']=test_df['Age'].fillna(test_df['Age'].median())

train_df['No_of_orders_placed']=train_df['No_of_orders_placed'].fill
na(train_df['No_of_orders_placed'].median())
test_df['No_of_orders_placed']=test_df['No_of_orders_placed'].fillna
(test_df['No_of_orders_placed'].median())

train_df['Kid’s_Clothing']=train_df['Kid’s_Clothing'].fillna(train_d
f['Kid’s_Clothing'].median())
test_df['Kid’s_Clothing']=test_df['Kid’s_Clothing'].fillna(test_df['
Kid’s_Clothing'].median())

train_df['Home_&_Living']=train_df['Home_&_Living'].fillna(train_df[
'Home_&_Living'].median())
test_df['Home_&_Living']=test_df['Home_&_Living'].fillna(test_df['Ho
me_&_Living'].median())
```

3- Data standardization
We scale the data using StandardScaler():

```
scaler = StandardScaler().fit(train_df[["Age",
          "No_of_orders_placed",
          "is_premium_member",
          "Women’s_Clothing",
          "Men’s_Clothing",
          "Kid’s_Clothing",
          "Home_&_Living",
          "Beauty",
          "Electronics"]])

train_df[["Age",
          "No_of_orders_placed",
          "is_premium_member",
          "Women’s_Clothing",
          "Men’s_Clothing",
          "Kid’s_Clothing",
          "Home_&_Living",
          "Beauty",
          "Electronics"]] = scaler.transform(train_df[["Age",
          "No_of_orders_placed",
          "is_premium_member",
          "Women’s_Clothing",
          "Men’s_Clothing",
          "Kid’s_Clothing",
          "Home_&_Living",
          "Beauty",
          "Electronics"]])

test_df[["Age",
          "No_of_orders_placed",
          "is_premium_member",
          "Women’s_Clothing",
          "Men’s_Clothing",
          "Kid’s_Clothing",
          "Home_&_Living",
          "Beauty",
          "Electronics"]] = scaler.transform(test_df[["Age",
          "No_of_orders_placed",
          "is_premium_member",
          "Women’s_Clothing",
          "Men’s_Clothing",
          "Kid’s_Clothing",
          "Home_&_Living",
          "Beauty",
          "Electronics"]])
```

4- Solving the problem of non-numeric data

In this way, we use dictionaries and also convert date from string to date-time and then to number so that they can be used in classification models.

```
Preferred_Theme_dict = {
    "New_UI": 1,
    "Old_UI": 2
}
train_df['Preferred_Theme'] = train_df['Preferred_Theme'].map(Prefer
red_Theme_dict)
test_df['Preferred_Theme'] = test_df['Preferred_Theme'].map(Preferre
d_Theme_dict)


Gender_dict = {
    'Male' : 1,
    'Female' : 2,
    'Not_Specified' : 3
}
train_df['Gender'] = train_df['Gender'].map(Gender_dict)
test_df['Gender'] = test_df['Gender'].map(Gender_dict)


State_dict = {
    'California' :            0,
    'British Columbia' :      1,
    'New South Wales' :       2,
    'Western Australia' :     3,
    'West Bengal' :           4,
    'Catalonia' :             5,
    'Bavaria' :               6,
    'Tamil Nadu' :            7,
    'Singapore' :             8,
    'Maharashtra' :           9,
    'Central Hungary' :       10,
    'New Delhi' :             11,
    'Ile-De-France' :         12,
    'Tokyo' :                 13,
    'Vienna' :                14,
    'Tuscany' :               15,
    'England' :               16,
    'Berlin' :                17,
    'New York' :              18,
    'Ontario' :               19,
    'Taiwan' :                20
```

```
}
train_df['State'] = train_df['State'].map(State_dict)
test_df['State'] = test_df['State'].map(State_dict)


import datetime
train_df['Sign_up_date'] = pd.to_datetime(train_df['Sign_up_date'],
errors='coerce')
test_df['Sign_up_date'] = pd.to_datetime(test_df['Sign_up_date'], er
rors='coerce')
train_df['Last_order_placed_date'] = pd.to_datetime(train_df['Last_o
rder_placed_date'], errors='coerce')
test_df['Last_order_placed_date'] = pd.to_datetime(test_df['Last_ord
er_placed_date'], errors='coerce')

train_df['Sign_up_date'] = train_df['Sign_up_date'].values.astype(fl
oat)
test_df['Sign_up_date'] = test_df['Sign_up_date'].values.astype(floa
t)
train_df['Last_order_placed_date'] = train_df['Last_order_placed_dat
e'].values.astype(float)
test_df['Last_order_placed_date'] = test_df['Last_order_placed_date'
].values.astype(float)
```

Correlation matrix for the problem target:

```
Age                         0.091663
Gender                      0.280818
State                       0.047900
No_of_orders_placed         0.190048
Sign_up_date               -0.010669
Last_order_placed_date     -0.066205
is_premium_member           0.216268
Women’s_Clothing            0.030801
Men’s_Clothing             -0.251957
Kid’s_Clothing              0.270812
Home_&_Living               0.203302
Beauty                      0.206975
Electronics                -0.051595
Preferred_Theme             1.000000
Name: Preferred_Theme, dtype: float64
```

5- K-Nearest Neighbor Algorithm:

In pattern recognition, K-Nearest Neighbor is a non-parametric statistical method used for statistical classification and regression. In both cases, Ki contains the closest training example in the data space and its output varies depending on the type used in classification and regression. In the classification mode, according to the specified value for ki, it calculates the distance of the point we want to label with the nearest points, and according to the maximum number of votes of these neighboring points, it makes a decision regarding the label of the desired point. we do. Different methods can be used to calculate this distance, one of the most prominent of these methods is the Euclidean distance. In the case of regression, the average of the values obtained from the key is its output. Since the calculations of this algorithm are based on distance, data normalization can help improve its performance.

1- The steps of the knn algorithm will include the following:
2- Load the data.
3- Choose K as the number of nearest neighbors.
4- For each of the primary data:
5- Calculate the distance between the data in question and each of the original data.
6- Add sample interval and endbands to a set.
7- Sort the set by distance from smallest to largest.
8- Select the K points of the first member of the sorted set.
9- Declare the output depending on the mode or classification mode.

```
from collections import Counter

import numpy as np


def euclidean_distance(x1, x2):
    return np.sqrt(np.sum((x1 - x2) ** 2))


class KNN:
    def __init__(self, k=3):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
```

```
    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        # Compute distances between x and all examples in the traini
ng set
        distances = [euclidean_distance(x, x_train) for x_train in s
elf.X_train]
        # Sort by distance and return indices of the first k neighbo
rs
        k_idx = np.argsort(distances)[: self.k]
        # Extract the labels of the k nearest neighbor training samp
les
        k_neighbor_labels = [self.y_train[i] for i in k_idx]
        # return the most common class label
        most_common = Counter(k_neighbor_labels).most_common(1)
        return most_common[0][0]


if __name__ == "__main__":
    # Imports
    from matplotlib.colors import ListedColormap
    from sklearn import datasets
    from sklearn.model_selection import train_test_split

    cmap = ListedColormap(["#FF0000", "#00FF00", "#0000FF"])

    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    iris = datasets.load_iris()
    X, y = iris.data, iris.target

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=1234
    )



clf = KNN(k=3)
clf.fit(X_train, y_train)
predictions = clf.predict(X_test)
```

```
print("KNN classification accuracy:", accuracy(y_test, predictions))
```

این الگوریتم به ازای k=3 دقتی حدود به 100 درصد دارد.

6- Checking the performance of support vector machines and linear regression

Using the available packages, we also implement SVM and regression on the data, and both achieve very high accuracy:

```
from sklearn.metrics import classification_report
from sklearn.metrics import mean_squared_error

print("==============Support Vector Machine==============")
from sklearn.svm import SVC
SVM = SVC(kernel = 'linear', random_state = 0)
SVM.fit(X_train, y_train)
predictions1 = SVM.predict(X_test)
print("SVM classification accuracy:", accuracy(y_test, predictions1)
)
print(classification_report(y_test, predictions1))

print("==============LinearRegression==============")
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
predictions2 = lin_reg.predict(X_test)
lin_mse = mean_squared_error(y_test, predictions2)
lin_rmse = np.sqrt(lin_mse)
print("lin_reg classification rmse:", accuracy(y_test, predictions2)
)
```

7-Evaluation

-Accuracy

Accuracy is one of the evaluation criteria of classification models. Informally, accuracy is the fraction of predictions that our model made correctly. Formally, accuracy has the following definition: accuracy = number of correct predictions total number of predictions

-Precision

One of the performance indicators of a machine learning model is the quality of a positive prediction made by the model. Accuracy refers to the number of true positives divided by the total number of positive predictions (ie, the number of true positives plus the number of false positives).

-Recall

Recall is literally how many true positives are remembered (found), that is, how many correct hits are also found. Accuracy (your formula is incorrect) is how many of the hits that were returned were true positives, that is, how many of the hits found were correct.

-F1-score

F1 score is one of the most important evaluation criteria in machine learning. This nicely summarizes the predictive performance of a model by combining two competing measures—precision and recall.

For our KNN model, we have achieved the following evaluation, which indicates 100% accuracy of our model.

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00         8

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```