ad-click-prediction

September 11, 2024

1 Understanding the Problem

[18]:

```
 
```

2 Load and Explore the Data

[19]:

```
import pandas as pd
import seaborn as sns

df = pd.read_csv('/content/ad_click_dataset.csv')
df.set_index('id', inplace = True)
df.head()
```

[19]:

```
     full_name   age      gender device_type ad_position browsing_history  \
id
670    User670  22.0         NaN     Desktop         Top         Shopping
3044  User3044   NaN        Male     Desktop         Top              NaN
5912  User5912  41.0  Non-Binary         NaN        Side        Education
5418  User5418  34.0        Male         NaN         NaN    Entertainment
9452  User9452  39.0  Non-Binary         NaN         NaN     Social Media

     time_of_day  click
id
670    Afternoon      1
3044         NaN      1
5912       Night      1
5418     Evening      1
9452     Morning      0
```

[20]:

```
df.shape
```

[20]:

```
(10000, 8)
```

[21]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10000 entries, 670 to 3056
```

```
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   full_name         10000 non-null  object
 1   age               5234 non-null   float64
 2   gender            5307 non-null   object
 3   device_type       8000 non-null   object
 4   ad_position       8000 non-null   object
 5   browsing_history  5218 non-null   object
 6   time_of_day       8000 non-null   object
 7   click             10000 non-null  int64
dtypes: float64(1), int64(1), object(6)
memory usage: 703.1+ KB
```

[22]:

```
df.describe()
```

[22]:

```
               age         click
count  5234.000000  10000.000000
mean     40.197363      0.650000
std      13.126420      0.476993
min      18.000000      0.000000
25%      29.000000      0.000000
50%      39.500000      1.000000
75%      52.000000      1.000000
max      64.000000      1.000000
```

[23]:

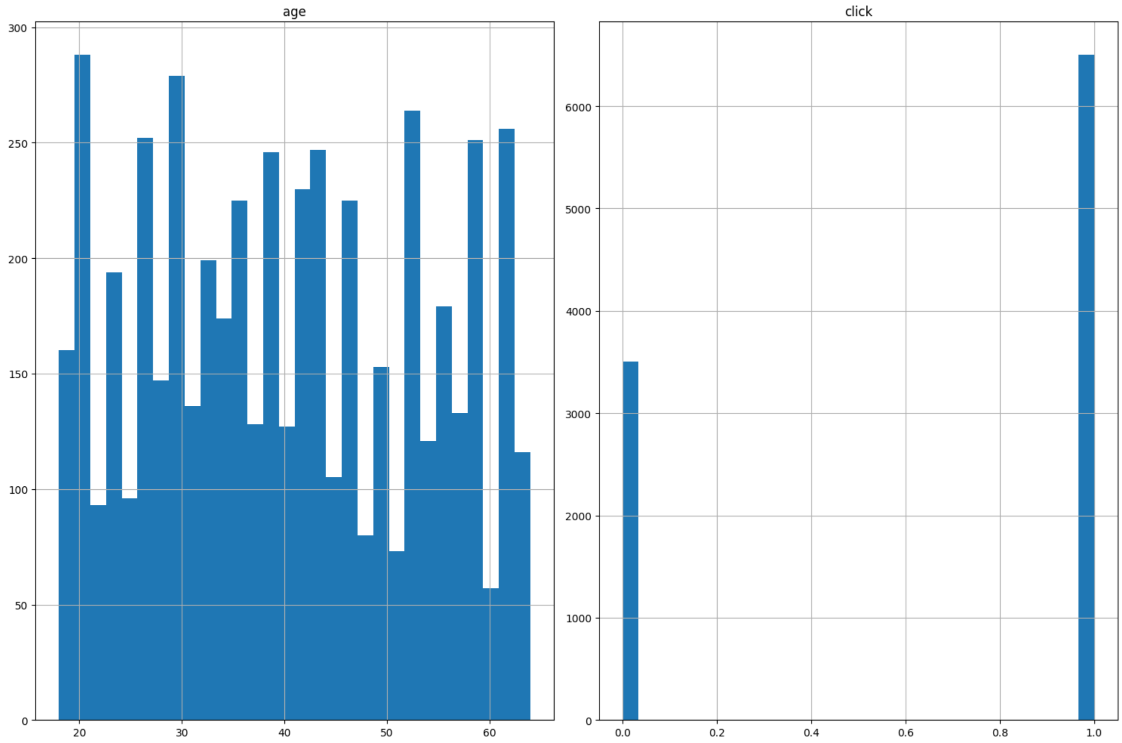
```
df.columns
```

[23]:

```
Index(['full_name', 'age', 'gender', 'device_type', 'ad_position',
       'browsing_history', 'time_of_day', 'click'],
      dtype='object')
```

[24]:

```
import matplotlib.pyplot as plt
df.hist(bins=30, figsize=(15, 10))
plt.tight_layout()
plt.show()
```



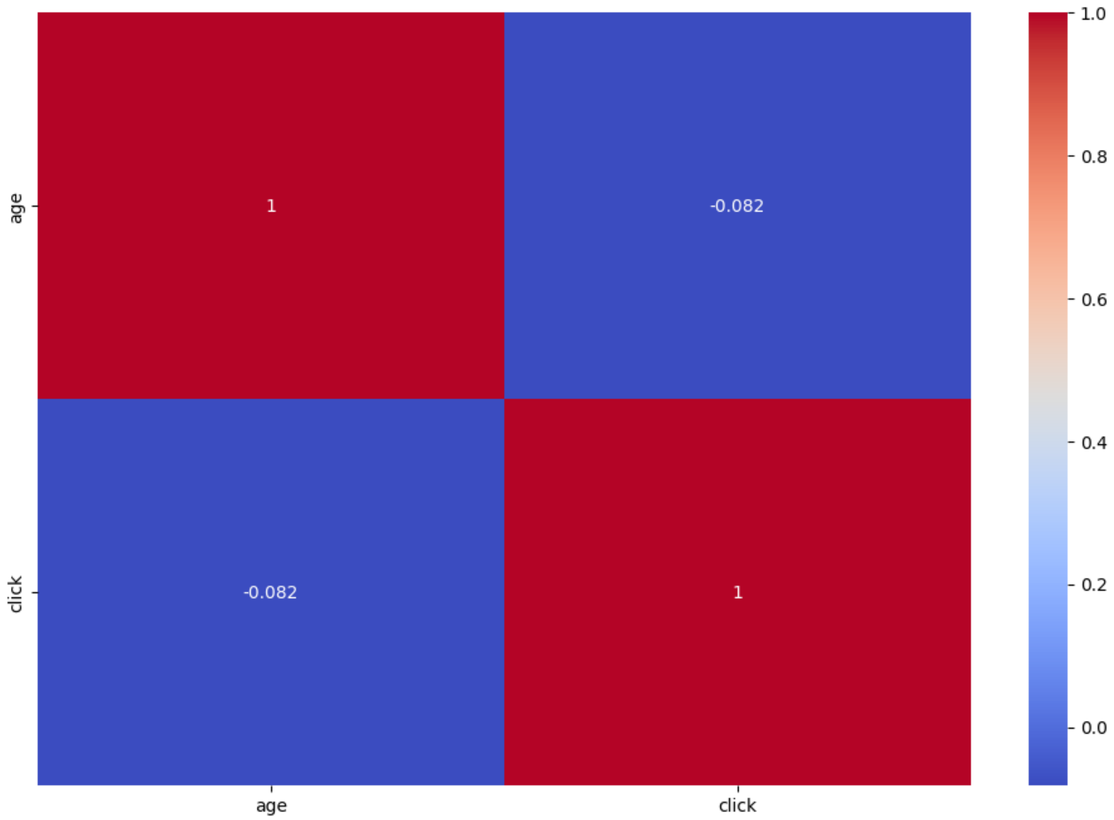
[25]:

```
corr_matrix = df.select_dtypes([int, float]).corr()
print(corr_matrix)
```

```
            age     click
age    1.000000 -0.082056
click -0.082056  1.000000
```

[26]:

```
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```



3 Data Preprocessing

3.1 Encode Categorical Variables

[27]:
```
from sklearn.preprocessing import LabelEncoder
obj_cols = df.select_dtypes(object).columns
for col in obj_cols:
  encoder = LabelEncoder()
  df[col] = encoder.fit_transform(df[col])

df.dtypes
```

[27]:
```
full_name              int64
age                  float64
gender                 int64
device_type            int64
ad_position            int64
browsing_history       int64
time_of_day            int64
click                  int64
```

```
dtype: object
```

3.2 Handling Missing Values

[28]:

```
df['age'].fillna(df['age'].mean(), inplace=True)
```

[29]:

```
df.isnull().sum()
```

[29]:

```
full_name           0
age                 0
gender              0
device_type         0
ad_position         0
browsing_history    0
time_of_day         0
click               0
dtype: int64
```

[30]:

```
import pandas as pd

duplicates = df.duplicated()

duplicates_in_subset = df.duplicated(subset=['full_name', 'age', 'gender',␣
 ↪'device_type', 'ad_position', 'browsing_history', 'time_of_day', 'click'])

# Remove all duplicate rows based on all columns
df_cleaned = df.drop_duplicates()

# Remove duplicates based on a subset of columns
df_cleaned = df.drop_duplicates(subset=['full_name', 'age', 'gender',␣
 ↪'device_type', 'ad_position', 'browsing_history', 'time_of_day', 'click'])

df_cleaned = df.groupby(['full_name', 'age', 'gender', 'device_type',␣
 ↪'ad_position', 'browsing_history', 'time_of_day']).agg({'click': 'sum'}).
 ↪reset_index()

print(df_cleaned.info())
print(df_cleaned.duplicated().sum())  # Should return 0 if all duplicates are␣
 ↪removed
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7147 entries, 0 to 7146
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   full_name         7147 non-null   int64
 1   age               7147 non-null   float64
 2   gender           7147 non-null   int64
 3   device_type      7147 non-null   int64
 4   ad_position      7147 non-null   int64
 5   browsing_history  7147 non-null   int64
 6   time_of_day      7147 non-null   int64
 7   click            7147 non-null   int64
dtypes: float64(1), int64(7)
memory usage: 446.8 KB
None
0
```

4 Split the Data

[31]:

```
x = df.drop(columns = ['click'])
y = df['click']
```

[32]:

```
from sklearn.model_selection import train_test_split

xtrain, xtest, ytrain, ytest = train_test_split(x, y, train_size = 0.8)
```

5 Model Selection

[33]:

```
from sklearn.metrics import accuracy_score

def eval_model(model, xtrain, ytrain, xtest, ytest):
  model.fit(xtrain, ytrain)
  trainpred = model.predict(xtrain)
  testpred = model.predict(xtest)
  return accuracy_score(ytrain, trainpred), accuracy_score(ytest, testpred)
```

[34]:

```
result = pd.DataFrame(columns = ['Model Number', 'Name', 'Training Accuracy',␣
  ↪'Testing Accuracy'])
result.head()
```

[34]:

```
Empty DataFrame
Columns: [Model Number, Name, Training Accuracy, Testing Accuracy]
Index: []
```

[35]:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
row = []
row.extend([1, 'Decision Tree'])
row.extend(eval_model(model, xtrain, ytrain, xtest, ytest))
result.loc[len(result.index)] = row
result.head()
```

[35]:

```
   Model Number           Name  Training Accuracy  Testing Accuracy
0             1  Decision Tree                1.0            0.8325
```

[36]:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
row = []
row.extend([2, 'KNN'])
row.extend(eval_model(model, xtrain, ytrain, xtest, ytest))
result.loc[len(result.index)] = row
result.head()
```

[36]:

```
   Model Number           Name  Training Accuracy  Testing Accuracy
0             1  Decision Tree           1.000000            0.8325
1             2            KNN           0.865125            0.8230
```

[37]:

```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
row = []
row.extend([3, 'Extra Trees'])
row.extend(eval_model(model, xtrain, ytrain, xtest, ytest))
result.loc[len(result.index)] = row
result.head()
```

[37]:

```
   Model Number           Name  Training Accuracy  Testing Accuracy
0             1  Decision Tree           1.000000            0.8325
1             2            KNN           0.865125            0.8230
2             3    Extra Trees           1.000000            0.7720
```

[38]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
row = []
row.extend([4, 'Logistic Regression'])
row.extend(eval_model(model, xtrain, ytrain, xtest, ytest))
result.loc[len(result.index)] = row
result.head()
```

[38]:

```
   Model Number                 Name  Training Accuracy  Testing Accuracy
0             1        Decision Tree           1.000000            0.8325
1             2                  KNN           0.865125            0.8230
2             3          Extra Trees           1.000000            0.7720
3             4  Logistic Regression           0.644875            0.6705
```

[39]:

```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
row = []
row.extend([5, 'Random Forest'])
row.extend(eval_model(model, xtrain, ytrain, xtest, ytest))
```

```
result.loc[len(result.index)] = row
result.head()
```

```
[39]:    Model Number                 Name  Training Accuracy  Testing Accuracy
      0             1        Decision Tree           1.000000            0.8325
      1             2                  KNN           0.865125            0.8230
      2             3          Extra Trees           1.000000            0.7720
      3             4  Logistic Regression           0.644875            0.6705
      4             5        Random Forest           1.000000            0.7990
```

```
[40]: from sklearn.svm import SVC
      model = SVC()
      row = []
      row.extend([6, 'SVM'])
      row.extend(eval_model(model, xtrain, ytrain, xtest, ytest))
      result.loc[len(result.index)] = row
      result.head()
```

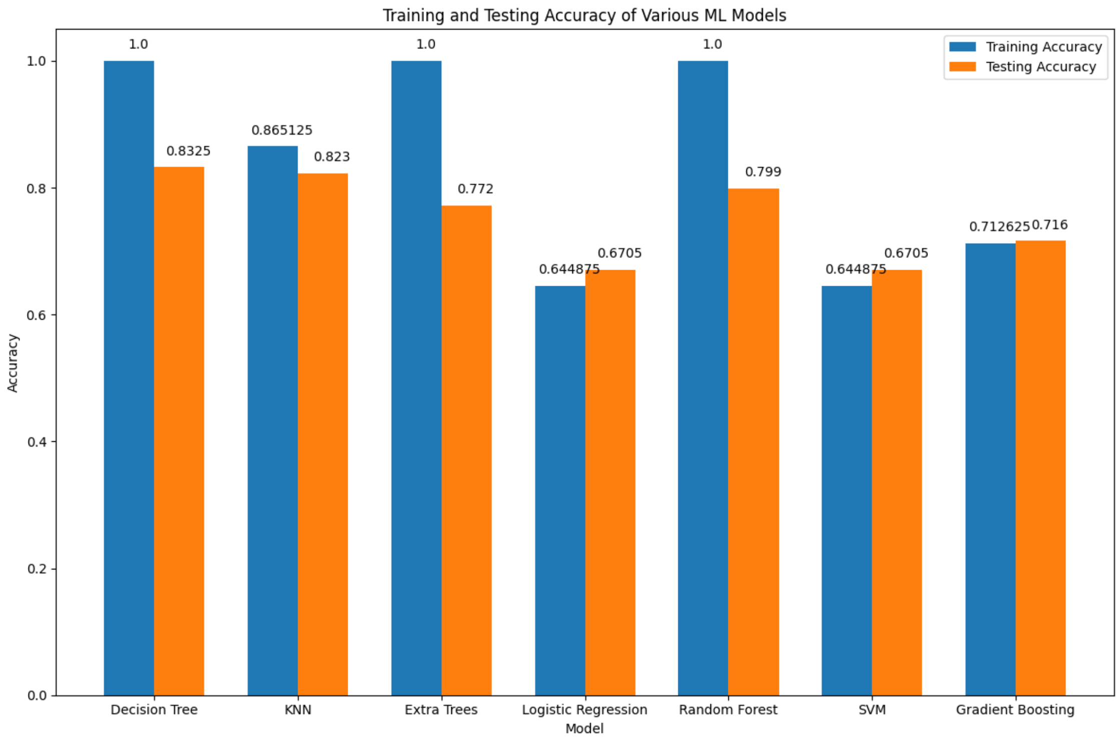
```
[40]:    Model Number                 Name  Training Accuracy  Testing Accuracy
      0             1        Decision Tree           1.000000            0.8325
      1             2                  KNN           0.865125            0.8230
      2             3          Extra Trees           1.000000            0.7720
      3             4  Logistic Regression           0.644875            0.6705
      4             5        Random Forest           1.000000            0.7990
```

```
[41]: from sklearn.ensemble import GradientBoostingClassifier
      model = GradientBoostingClassifier()
      row = []
      row.extend([7, 'Gradient Boosting'])
      row.extend(eval_model(model, xtrain, ytrain, xtest, ytest))
      result.loc[len(result.index)] = row
      result
```

```
[41]:    Model Number                 Name  Training Accuracy  Testing Accuracy
      0             1        Decision Tree           1.000000            0.8325
      1             2                  KNN           0.865125            0.8230
      2             3          Extra Trees           1.000000            0.7720
      3             4  Logistic Regression           0.644875            0.6705
      4             5        Random Forest           1.000000            0.7990
      5             6                  SVM           0.644875            0.6705
      6             7    Gradient Boosting           0.712625            0.7160
```

```
[42]: import numpy as np

      models = result['Name']
      training_accuracies = np.round(result['Training Accuracy'],7)
      testing_accuracies = np.round(result['Testing Accuracy'],7)
```

```
x = np.arange(len(models))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots(figsize=(12, 8))
rects1 = ax.bar(x - width/2, training_accuracies, width, label='Training␣
 ↪Accuracy')
rects2 = ax.bar(x + width/2, testing_accuracies, width, label='Testing␣
 ↪Accuracy')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Model')
ax.set_ylabel('Accuracy')
ax.set_title('Training and Testing Accuracy of Various ML Models')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

# Function to add labels on top of the bars
def autolabel(rects):
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(7, 7),  # 7 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

autolabel(rects1)
autolabel(rects2)

fig.tight_layout()

plt.show()
```



6 Model Training

[43]:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(xtrain, ytrain)
```

[43]:

```
KNeighborsClassifier()
```

7 Model Evaluation

[44]:

```
trainpred = model.predict(xtrain)
testpred = model.predict(xtest)
```

[45]:

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

[46]:

```
print(classification_report(ytrain, trainpred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.65      0.77      2841
           1       0.84      0.98      0.90      5159
```

```
    accuracy                           0.87      8000
   macro avg       0.90      0.82      0.84      8000
weighted avg       0.88      0.87      0.86      8000
```

[47]:
```
print(classification_report(ytest, testpred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.53      0.66       659
           1       0.81      0.97      0.88      1341

    accuracy                           0.82      2000
   macro avg       0.85      0.75      0.77      2000
weighted avg       0.83      0.82      0.81      2000
```