Conversion Rate Prediction

A case study by:

U Aaditya - CB.EN.U4CSE18402

K V Phanindra Reddy – CB.EN.U4CSE18433

N S S S Rohit Kallakuri – CB.EN.U4CSE18449

Contents

Introduction 3
The Problem Statement 3
What Is Conversion Rate, And Why Is It Important? 3
Aim 3
The Data 4
Tech Stack 5
Machine Learning Algorithm Used 5
Pre-processing steps to be followed: 5
Hypothesis 5
Code: 6
Preprocessing: 6
Visualizations: 7
Training the models: 10
Optimizing the models: ..... **Error! Bookmark not defined.**
Evaluating the models: ..... **Error! Bookmark not defined.**
Conclusion: 14

Introduction

We are being increasingly exposed to the idea that data is the fuel of the 21st century, and it is rightfully so, because access to data can give us insights like never before. One such use of data is in conversion rate prediction and optimization.

The Problem Statement

Given data on the details of users visiting a certain website, build a model that predicts if a specific user would decide to go ahead and convert into a potential customer.

What Is Conversion Rate, And Why Is It Important?

In a general sense, conversion rate is the percentage of visitors of a website that complete a desired goal among the total number of visitors.

A high conversion rate means that there is a higher percentage of visitors that get converted into a customer, and it is indicative of successful web design and marketing, and insights into conversion rate can help the company optimize marketing and/or web design, which makes it a tool that can be used to gain a competitive edge.

Aim

The aim of this case study is to explore the application of machine learning on conversion rate prediction and optimization using different classification algorithms and analyzing the performance metrics in each one of them to determine the best algorithm for this given use case, and to deploy the algorithms on a web environment using Flask

The Data

The dataset we are using is a table called ‘conversion_data’. The description of the table is as follows:

Column	Description
country	User country based on IP address
age	User age as specified in sign-in
new_user	Specifies whether user created the account in this session or if they are a returning user
source	Specifies the marketing channel source
Ads	Specifies whether the user came to the site by clicking on an ad
Seo	Specifies whether the user came to the site by clicking on search results
Direct	Specifies whether the user came to the site by directly typing the URL
total_pages_visited	Specifies the count of the number of pages visited during the session
converted	The label that denotes whether the given user has converted. $converted = \begin{cases} 1\ if\ the\ user\ has\ converted \\ 0\ if\ the\ user\ hasn't\ converted \end{cases}$

Tech Stack

1. numpy – to handle numpy arrays
2. pandas – to handle and manipulate dataframes
3. sklearn – to preprocess the data and build the model
4. matplotlib and seaborn – to make plots
5. pickle – to export the models
6. Flask – to deploy the models

Machine Learning Algorithm Used

We are going to implement this model using the classification models given below:

1. Random Forest Classifier
2. K-Nearest Neighbors
3. Naïve Bayes
4. Support Vector Classifier

This will give us enough metrics to analyze and determine which algorithm is best for the given use case.

Pre-processing steps to be followed:

1. Removing unrealistic ages
2. Removing outliers
3. One Hot Encoding
4. Label Encoding

Hypothesis

We hypothesize that the Naïve Bayes algorithm will outperform the other models, and the reasoning behind it is that the nature of conversion in itself is probabilistic – we cannot predict that a visitor is going to get converted into a customer for sure based on what they do on a website.

Code:

Preprocessing:

There are some fake values for age here, as made evident by the max value of age at 123. Let's see how many unrealistic ages are there:

In [5]:
```
sorted(df['age'],reverse=True)[:25]
```

Out[5]:
```
[123,
 111,
 79,
 77,
 73,
 72,
 70,
 70,
 69,
 69,
 69,
 68,
 68,
 68,
 68,
 68,
 67,
 67,
 67,
 67,
 67,
 66,
 66,
 66,
 66]
```

Luckily, we only have 2 unrealistic values, and they can be removed easily. Let' get rid of them now:

In [6]:
```
print(df.loc[df['age']==123])
print(df.loc[df['age']==111])
```

```
       country  age  new_user source  total_pages_visited  converted
90928  Germany  123         0    Seo                   15          1
       country  age  new_user source  total_pages_visited  converted
295581      UK  111         0    Ads                   10          1
```

In [7]:
```
df.drop([90928,295581],inplace=True)
```

Now, let's see if the dataset has any null values:

In [8]:
```
df.isna().sum()
```

Out[8]:
```
country                0
age                    0
new_user               0
source                 0
total_pages_visited    0
converted              0
dtype: int64
```

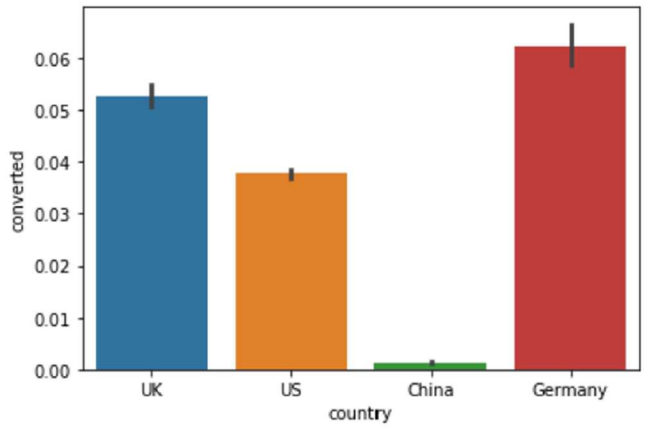
Luckily, we don't have any null values.

Visualizations:

Now, let's try to visualize the dataset. Let's see the demographic of the people who have converted:

In [9]:
```
sns.barplot(x='country',y='converted',data=df)
```

Out[9]:
```
<AxesSubplot:xlabel='country', ylabel='converted'>
```



With this, we can infer that the conversion rate in China is way lower than that in countries like the USA, UK and Germany.

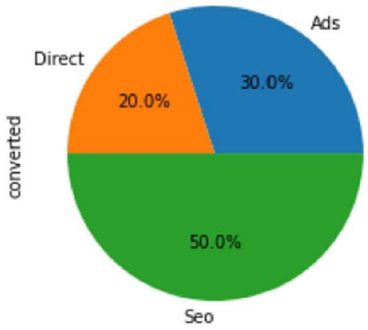
Now let's take a look at the proportion of people getting converted from each source

In [10]:
```
pie_data=df.groupby('source')['converted'].sum()
pie_data
```

Out[10]:
```
source
Ads       3059
Direct    2040
Seo       5099
Name: converted, dtype: int64
```

In [11]:
```
pie_data.plot.pie(autopct="%.1f%%")
```

Out[11]: <AxesSubplot:ylabel='converted'>

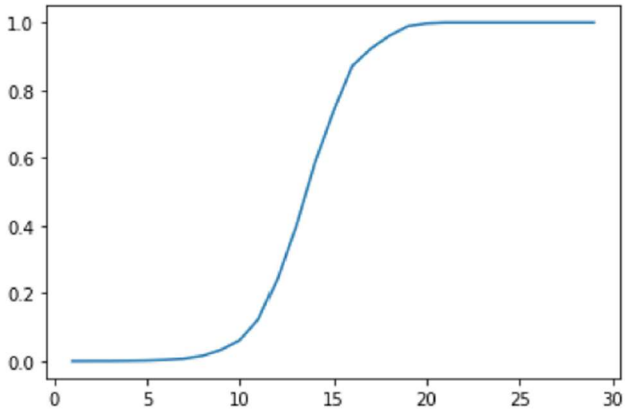


From this, we can infer that roughly 50% of the conversions happen through SEO marketing.

Now let's try to visualize the trend in conversion rate with number of pages visited:

In [12]:
```
data_pages = df.groupby('total_pages_visited')[['converted']].mean()
data_ages = df.groupby('age')[['converted']].mean()
plt.plot(data_pages.index,data_pages['converted'])
```

Out[12]: [<matplotlib.lines.Line2D at 0x1ca215b6a88>]

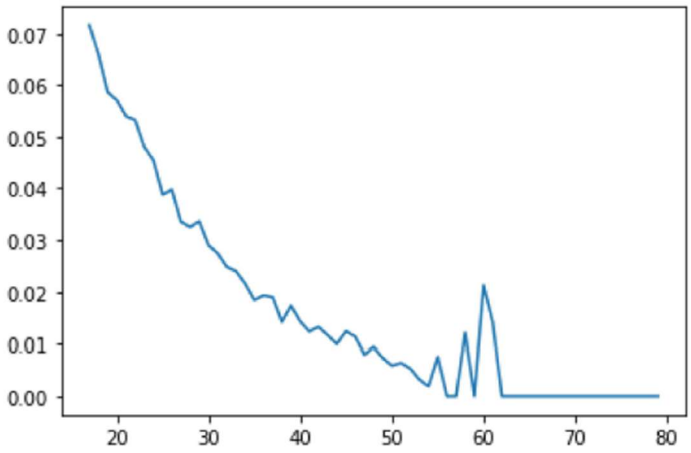


The x axis here is number of pages visited, and the y axis is conversion rate. We can see that as the number of pages increases, the conversion rate increases too

Now, let's try to visualize the effect of age on conversion rate:

In [13]:
```
plt.plot(data_ages.index,data_ages['converted'])
```

Out[13]:
```
[<matplotlib.lines.Line2D at 0x1ca21623c88>]
```



This shows a inverse proportionality between age and conversion rate - the higher the age, the lower the conversion.

Training the models:

Training the Random Forest Classifier

Now let's train a random forest classifier. For that, let's make a copy of the original dataframe, apply one hot encoding on the countries and source, split the dataset and train the model:

In [14]:
```
df2=df
df_country = pd.get_dummies(df2['country'])
df2 = pd.concat([df2, df_country], axis=1)
df_source = pd.get_dummies(df2['source'])
df2 = pd.concat([df2, df_source], axis=1)
df2.head()
```

Out[14]:

	country	age	new_user	source	total_pages_visited	converted	China	Germany	UK	US	Ads	Dire
0	UK	25	**1**	Ads	**1**	0	0	0	1	0	1	
1	US	23	**1**	Seo	5	0	0	0	0	**1**	0	
2	US	28	**1**	Seo	4	0	0	0	0	**1**	0	
3	China	39	**1**	Seo	5	0	1	0	0	0	0	
4	US	30	**1**	Seo	6	0	0	0	0	**1**	0	

In [15]:
```
X_train, X_test, y_train, y_test = train_test_split(df2[['total_pages_visited','China',
```

Now that we have split the dataset, let's train the model:

In [16]:
```
from sklearn.ensemble import RandomForestClassifier
forest = RandomForestClassifier(random_state=42)
forest.fit(X_train, np.array(y_train))
```

Out[16]:
```
RandomForestClassifier(random_state=42)
```

In [17]:
```
print('The accuracy of the random forest classifier model is',str(forest.score(X_test,y
```

```
The accuracy of the random forest classifier model is 98.50209878672877%
```

In [18]:
```
print('The baseline accuracy is '+str(100*(1-sum(df2['converted'])/len(df2['converted']
```

```
The baseline accuracy is 96.77480565974484%
```

Training the KNN classifier:

Now let's train the KNN classifier. For that, let's encode the labels and then split the model and train it:

In [19]:
```
from sklearn.preprocessing import LabelEncoder
df3=df
le=LabelEncoder()
df3['country']=le.fit_transform(df3['country'])
df3['source']=le.fit_transform(df3['source'])
```

In [20]:
```
df3.head()
```

Out[20]:

	country	age	new_user	source	total_pages_visited	converted
0	2	25	1	0	1	0
1	3	23	1	2	5	0
2	3	28	1	2	4	0
3	0	39	1	2	5	0
4	3	30	1	2	6	0

LabelEncoder encodes the data in ascending alphabetical order like this:

Country: 0-China, 1-Germany, 2-UK, 3-US Source: 0-Ads, 1-Direct, 2-SEO

Now let's split the dataset:

In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(df3[['total_pages_visited','country
```

In [23]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_model = KNeighborsClassifier(n_neighbors=3)

knn_model.fit(X_train,np.array(y_train))
```

Out[23]:
```
KNeighborsClassifier(n_neighbors=3)
```

In [24]:
```
print('The accuracy of the KNN model is '+str(100*knn_model.score(X_test,y_test))+'%')
```

```
The accuracy of the KNN model is 98.4388476798344%
```

Training the Naive Bayes classifier:

Now let's train the Naive Bayes classifier. For this we will be using Gaussian NB model in sklearn while also reusing the label encoded training and testing data used in KNN:

In [25]:
```
from sklearn.naive_bayes import GaussianNB

nb_model=GaussianNB()
nb_model.fit(X_train,np.array(y_train))
```

Out[25]:
```
GaussianNB()
```

In [26]:
```
print('The accuracy of the Naive Bayes model is '+str(100*nb_model.score(X_test,y_test)
```

```
The accuracy of the Naive Bayes model is 98.38613842408908%
```

Training the Support Vector classifier:

Now let's train the support vector classifier. For this we will be using the same label encoded training and testing data used in KNN:

In [27]:
```
from sklearn.svm import SVC

svc_model=SVC(kernel='linear')
svc_model.fit(X_train,np.array(y_train))
```

Out[27]:
```
SVC(kernel='linear')
```

In [28]:
```
print('The accuracy of the SVC model is '+str(100*svc_model.score(X_test,y_test)))
```

```
The accuracy of the SVC model is 98.6190174994729
```

In [29]:
```
import pickle
pickle.dump(svc_model, open('svc_model.pkl','wb'))
pickle.dump(nb_model, open('nb_model.pkl','wb'))
```



```
pickle.dump(knn_model, open('knn_model.pkl','wb'))
pickle.dump(forest, open('forest_model.pkl','wb'))
```

Deploying the models on a Flask server:

The models, in separate pickle files, were then deployed on a Flask server with the following code:

```
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

app = Flask(__name__)
knn_model = pickle.load(open('knn_model.pkl', 'rb'))
nb_model = pickle.load(open('nb_model.pkl', 'rb'))
svc_model = pickle.load(open('svc_model.pkl', 'rb'))
forest_model = pickle.load(open('forest_model.pkl', 'rb'))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict-knn',methods=['POST'])
def predict_knn():

    int_features = [int(x) for x in request.form.values()]
    print(int_features)
    final_features = [np.array(int_features)]
    print(final_features)
    prediction = knn_model.predict(final_features)

    print(prediction)

    return render_template('index.html', prediction_text='Conversion: {}'.format(
prediction))

@app.route('/predict-nb',methods=['POST'])
def predict_nb():

    int_features = [int(x) for x in request.form.values()]
    print(int_features)
    final_features = [np.array(int_features)]
    print(final_features)
    prediction = nb_model.predict(final_features)

    print(prediction)
```

```
    return render_template('index.html', prediction_text='Conversion: {}'.format(
prediction))

@app.route('/predict-svc',methods=['POST'])
def predict_svc():

    int_features = [int(x) for x in request.form.values()]
    print(int_features)
    final_features = [np.array(int_features)]
    print(final_features)
    prediction = svc_model.predict(final_features)

    print(prediction)

    return render_template('index.html', prediction_text='Conversion: {}'.format(
prediction))

@app.route('/results',methods=['POST'])
def results():

    data = request.get_json(force=True)
    prediction = model.predict([np.array(list(data.values()))])

    output = prediction[0]
    return jsonify(output)

if __name__ == "__main__":
    app.run(debug=True)
```

All code can be found on GitHub, with the link in the Appendix subheading.

Conclusion:

Through this case study, we have explored the fitting and evaluation of the 4 algorithms specified above, and have also learnt how to deploy these algorithms on a web environment using Flask.

Findings:

From the models, we can conclude that the performance of all the models are somewhat similar, but given that the number of testing samples are in six figures, a decimal change in accuracy can definitely go a long way, and the Support Vector Classifier performs better than the other algorithms. It is a whole 0.1% higher than the second highest accuracy score, showing the effectiveness of the algorithm given the scale and the use case.

The confusion matrices of the algorithms are given below:

```
In [33]: confusionmatrixforest
Out[33]: array([[100488,    541],
                [  1022,   2295]], dtype=int64)

In [34]: confusionmatrixknn
Out[34]: array([[100463,    566],
                [  1063,   2254]], dtype=int64)

In [35]: confusionmatrixnb
Out[35]: array([[100079,    950],
                [   734,   2583]], dtype=int64)

In [36]: confusionmatrixsvc
Out[36]: array([[100684,    345],
                [  1096,   2221]], dtype=int64)
```

From this we can see that SVC does better with true negatives but doesn't do as well with true positives, while Naïve Bayes does better with true positives and false negatives but doesn't do as well with false positives and true negatives, and this might partly have to do with the probabilistic nature of the use case, implying that it does much better with predicting a conversion than it does with predicting a case without conversion.

Limitations:

The main limitation of this case study is that it uses a dataset with way more datasets with label converted=0 than with label converted=1. This has to do with the fact that conversion rates in general are very low, but this makes conversion very hard to predict.

Appendix:

GitHub link: aaditya47/Conversion-Prediction (github.com)