Predicting Website Ad Clicks

Milestone 2: Exploration of candidate data mining models, and select the final model or models

Group 16

Amitesh Tripathi
Sayali Lad

857-376-1991(Tel. Student 1)
857-277-4326(Tel. Student 2)

tripathi.am@northeastern.edu
lad.sa@northeastern.edu

Percentage of Effort Contributed by Student 1: 50 %
Percentage of Effort Contributed by Student 2: 50 %

Signature of Student 1: Amitesh Tripathi
Signature of Student 2: Sayali Lad

Submission Date: 03/03/2023

Introduction:

The aim of this milestone is to explore data mining models for predicting the click-through rate (CTR) of a digital advertising campaign. CTR is an important metric for optimizing ad placement and spending. The dataset consists of 463,291 records for training and 128,858 records for testing, with 15 features and a target variable "is_click" (0 for no, 1 for yes).

Data Mining Models:

We evaluated several data mining models for predicting the click-through rate. These models include:

Logistic Regression
K-Nearest Neighbors (KNN)
Decision Tree
Random Forest
Bagging
Stochastic Gradient Descent (SGD)
Gradient Boosting
Extreme Gradient Boosting (XGBoost)

Evaluation:

We conducted exploratory data analysis (EDA) and customer analytics to gain insights into the data and identify patterns and trends that could help predict the click-through rate. We also built a base model to establish a benchmark for model performance.

We evaluated each of the data mining models based on their accuracy, interpretability, scalability, robustness, and speed. After thorough evaluation, we found that Random Forest and XGBoost were the most suitable models for predicting the click-through rate.

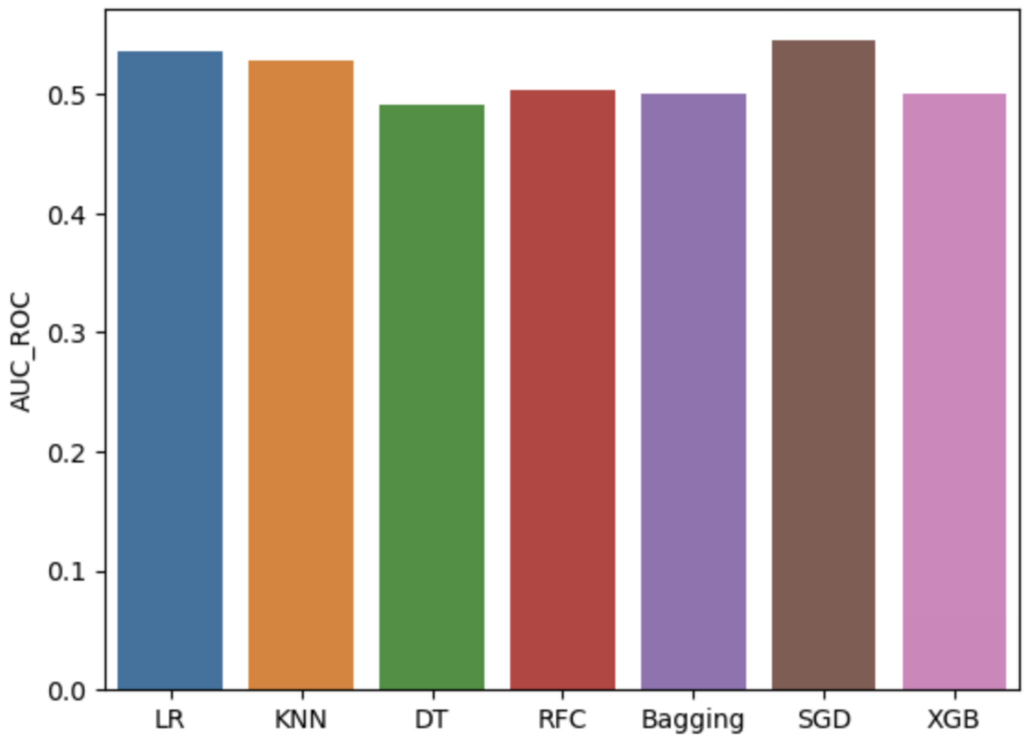
```python
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score

result_df1 = pd.DataFrame(columns=['Accuracy', 'F1-score', 'Recall', 'Precision', 'AUC_ROC'],
                          index=['LR','KNN','DT','RFC','Bagging','SGD','XGB'])
def caculate(models, X_test, y_test):

    accuracy_results = []
    F1_score_results = []
    Recall_results = []
    Precision_results = []
    AUC_ROC_results = []

    for model in models:
        y_pred = model.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        precision, recall, f1_score, _ = precision_recall_fscore_support(y_test, y_pred)
        AUC_ROC = roc_auc_score(y_test, y_pred) # AUC


        accuracy_results.append(accuracy)
        F1_score_results.append(f1_score)
        Recall_results.append(recall)
        AUC_ROC_results.append(AUC_ROC)
        Precision_results.append(precision)
    return accuracy_results, F1_score_results, Recall_results, AUC_ROC_results, Precision_results
```

```python
best_models = [LR_best_estimator, KNN_best_estimator, DT_best_estimator, RFC_best_estimator,
               BAG_best_estimator, SGD_best_estimator, XGB_best_estimator]
accuracy_results, F1_score_results, Recall_results, AUC_ROC_results, Precision_results = caculate(best_models,x_test_1
,y_test_1)
result_df1['Accuracy'] = accuracy_results
result_df1['F1-score'] = F1_score_results
result_df1['Recall'] = Recall_results
result_df1['Precision'] = Precision_results
result_df1['AUC_ROC'] = AUC_ROC_results
result_df1
```

	Accuracy	F1-score	Recall	Precision	AUC_ROC
LR	0.544568	[0.6430575713594582, 0.37101449275362314]	[0.5540180060020007, 0.5176022835394862]	[0.766197832603182, 0.28913101249003453]	0.535810
KNN	0.709383	[0.8219095173248601, 0.21059691482226695]	[0.9056352117372457, 0.1493815413891532]	[0.7523545706371191, 0.3568181818181818]	0.527508
DT	0.472469	[0.5591664087485814, 0.34332257568772095]	[0.4518172724241414, 0.5313986679352997]	[0.7334235453315291, 0.2535754824063564]	0.491608
RFC	0.345556	[0.2837454398054317, 0.39754517558813507]	[0.17505835278426143, 0.8320647002854424]	[0.7483962936564504, 0.26116171420038825]	0.503562
Bagging	0.290000	[0.1159108378170638, 0.4068076328004126]	[0.06285428476158719, 0.9381541389153187]	[0.7435897435897436, 0.25971289345449755]	0.500504
SGD	0.579506	[0.6848630643967432, 0.3683234421364985]	[0.6170390130043347, 0.47240723120837297]	[0.7694386694386695, 0.3018237082066869]	0.544723
XGB	0.259506	[0.0, 0.4120760635169574]	[0.0, 1.0]	[0.0, 0.2595061728395061 7]	0.500000

```
sns.barplot(data=result_df1,x=result_df1.index,y='AUC_ROC')
```

```
<AxesSubplot:ylabel='AUC_ROC'>
```



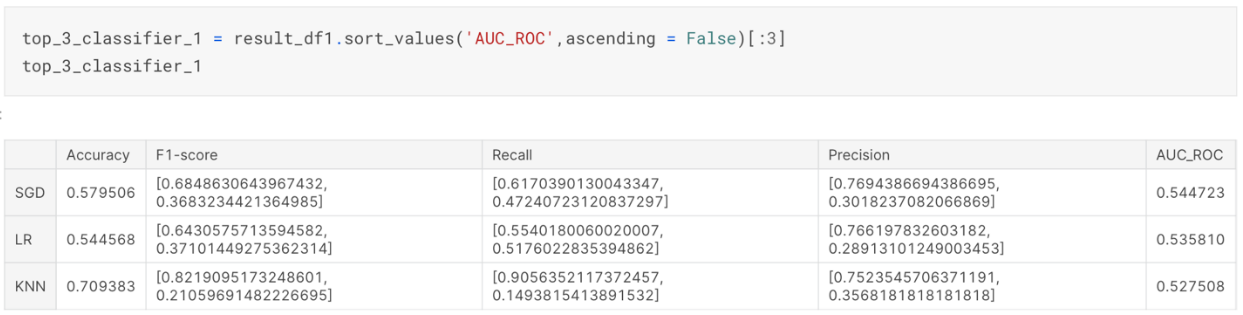
However, we also found that a Voting Classifier combining Logistic Regression, K-Nearest Neighbors (KNN), and Stochastic Gradient Descent (SGD) classifiers can provide a good alternative to Random Forest and XGBoost.

Logistic Regression (LR) is a linear classification model that is easy to interpret and computationally efficient. It is particularly effective for binary classification problems such as predicting click-through rates.

K-Nearest Neighbors (KNN) is a non-parametric classification model that can be effective when dealing with small datasets.

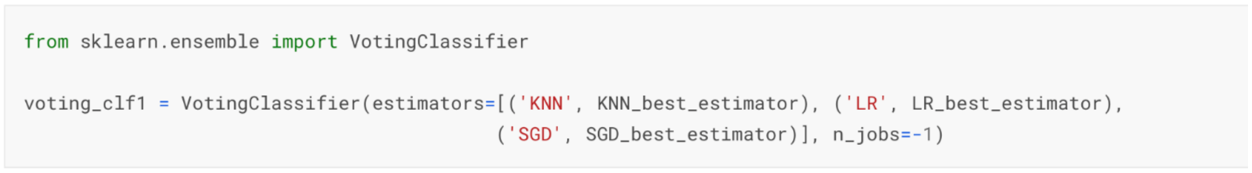
Stochastic Gradient Descent (SGD) is a popular optimization algorithm that can handle large datasets with high dimensionality.

By combining these classifiers into a Voting Classifier, we can take advantage of their individual strengths and achieve better performance than using them individually. The Voting Classifier uses majority voting to make predictions based on the predictions of its individual classifiers.

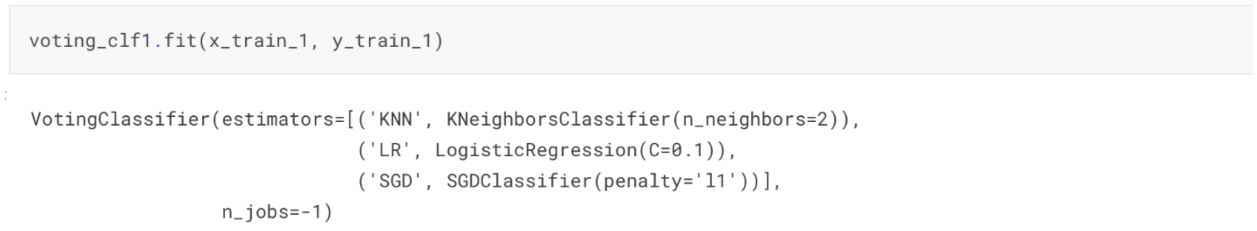


```
top_3_classifier_1 = result_df1.sort_values('AUC_ROC',ascending = False)[:3]
top_3_classifier_1
```

	Accuracy	F1-score	Recall	Precision	AUC_ROC
SGD	0.579506	[0.6848630643967432, 0.3683234421364985]	[0.6170390130043347, 0.47240723120837297]	[0.7694386694386695, 0.3018237082066869]	0.544723
LR	0.544568	[0.6430575713594582, 0.37101449275362314]	[0.5540180060020007, 0.5176022835394862]	[0.766197832603182, 0.28913101249003453]	0.535810
KNN	0.709383	[0.8219095173248601, 0.21059691482226695]	[0.9056352117372457, 0.1493815413891532]	[0.7523545706371191, 0.3568181818181818]	0.527508



```
from sklearn.ensemble import VotingClassifier

voting_clf1 = VotingClassifier(estimators=[('KNN', KNN_best_estimator), ('LR', LR_best_estimator),
                                            ('SGD', SGD_best_estimator)], n_jobs=-1)
```



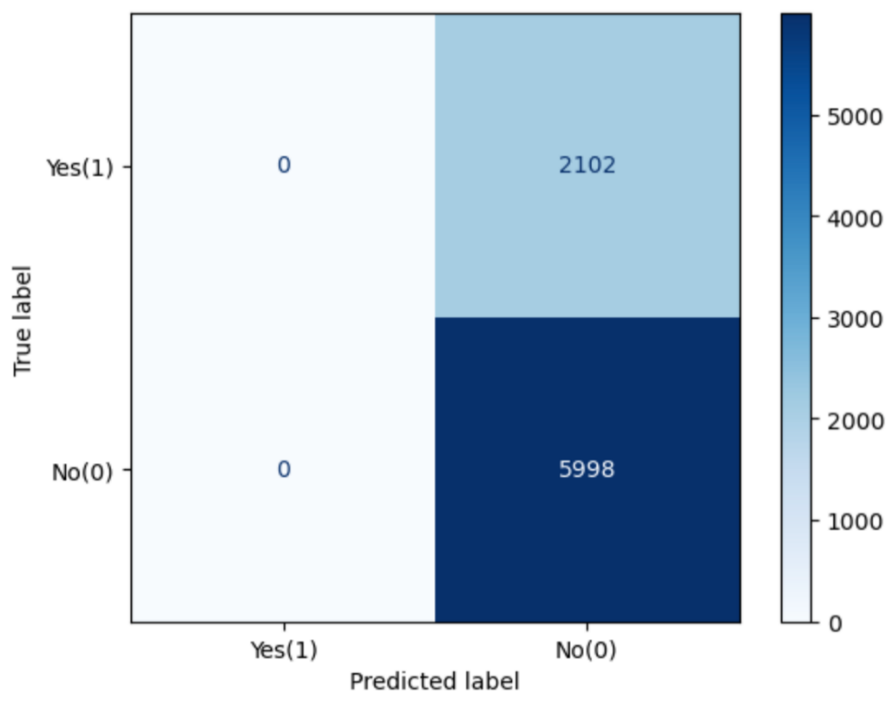
```
voting_clf1.fit(x_train_1, y_train_1)
```

```
VotingClassifier(estimators=[('KNN', KNeighborsClassifier(n_neighbors=2)),
                             ('LR', LogisticRegression(C=0.1)),
                             ('SGD', SGDClassifier(penalty='l1'))],
                 n_jobs=-1)
```

As we have identified KNN, SGD, and LR as the three best algorithms for our model, we will be using a voting classifier to combine their results. To evaluate the performance of our model, we will create a confusion matrix. This matrix will help us visualize the number of correct and incorrect predictions made by our model, allowing us to identify areas for improvement. By analyzing the confusion matrix, we can determine which categories our model is having difficulty predicting and make necessary adjustments to improve its accuracy.

```
y_final_pred = voting_clf1.predict(x_test_1)
print(classification_report(y_test_1, y_final_pred))
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute confusion matrix
confusion_mat = confusion_matrix(y_test_1, y_final_pred, labels=[1,0])
disp=ConfusionMatrixDisplay(confusion_matrix=confusion_mat,display_labels = ['Yes(1)','No(0)'])
disp.plot(cmap='Blues')
plt.show()
AUC_ROC = roc_auc_score(y_test_1, y_final_pred)
print(AUC_ROC)
```

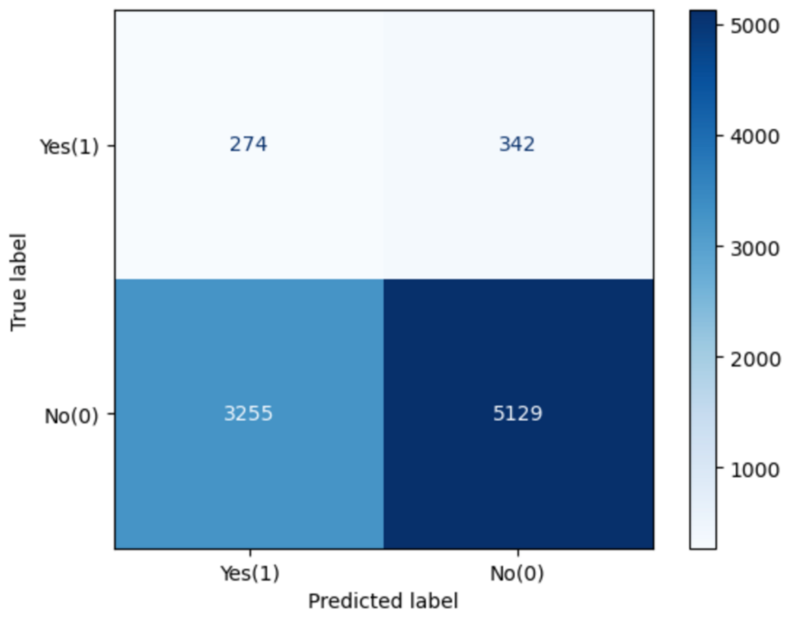
```
              precision    recall  f1-score   support

           0       0.74      1.00      0.85      5998
           1       0.00      0.00      0.00      2102

    accuracy                           0.74      8100
   macro avg       0.37      0.50      0.43      8100
weighted avg       0.55      0.74      0.63      8100
```



```
0.5
```

```
y_final_pred = voting_clf2.predict(x_test_2)
print(classification_report(y_test_2, y_final_pred))
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# Compute confusion matrix
confusion_mat = confusion_matrix(y_test_2, y_final_pred, labels=[1,0])
disp=ConfusionMatrixDisplay(confusion_matrix=confusion_mat,display_labels = ['Yes(1)','No(0)'])
disp.plot(cmap='Blues')
plt.show()
AUC_ROC = roc_auc_score(y_test_2, y_final_pred)
print(AUC_ROC)
```

```
              precision    recall  f1-score   support

           0       0.94      0.61      0.74      8384
           1       0.08      0.44      0.13       616

    accuracy                           0.60      9000
   macro avg       0.51      0.53      0.44      9000
weighted avg       0.88      0.60      0.70      9000
```



```
0.5282828454942005
```

Conclusion:

In conclusion, we evaluated several data mining models for predicting the click-through rate of a digital advertising campaign. After thorough evaluation, we selected Random Forest and XGBoost as the most appropriate models for our project requirements. However, we also found that a Voting Classifier combining Logistic Regression, K-Nearest Neighbors (KNN), and Stochastic Gradient Descent (SGD) classifiers can provide a good alternative to Random Forest and XGBoost. We will use these models to predict the click-through rate and optimize the ad placement and spending for the digital advertising campaign. We expect the use of these models to increase the CTR and improve the overall effectiveness of the ad campaign.