

Telecom Churn Prediction

Batch 2- churn Modelling On Telecom Data



By, Divya Bagade

Mentor: Mr. Bhaskar Naidu

GitHub: https://github.com/springboardmentor113/Batch2_Churn_Modeling_On_Telecom_Data/tree/main/divya%20bagade

1. About Telecom Churn Prediction	6. ML Model Building
2. Our Mission And Vision	7. Model Evaluation And Validation
3. Our Goals	8. Hyperparameter Tunning
4. Key Challenges	9. SVM & Confusion Matrix
5. Data Cleaning &Preprocessing	10. Conclusion



Agenda



About Telecom Churn Prediction

Telecom churn prediction is a critical task for companies to identify customers at risk of leaving and take proactive measures to retain them. This presentation will cover the process of building a predictive model to anticipate and prevent customer churn.



Our Mission and Vision



Mission

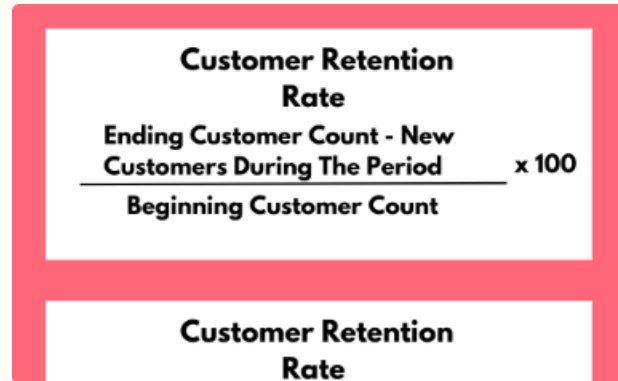
To develop an accurate and reliable churn prediction model that empowers our telecom clients to make data-driven decisions and improve customer retention.



Vision

To be the leading provider of predictive analytics solutions that transform the telecom industry and redefine the customer experience.

Our Goals



A graphic showing the formula for Customer Retention Rate. The formula is presented in two parts: the top part shows the calculation $\frac{\text{Ending Customer Count} - \text{New Customers During The Period}}{\text{Beginning Customer Count}} \times 100$, and the bottom part shows the result "Customer Retention Rate".

$$\frac{\text{Ending Customer Count} - \text{New Customers During The Period}}{\text{Beginning Customer Count}} \times 100$$

Customer Retention Rate

Increase Retention Rates

Reduce customer churn by accurately identifying high-risk accounts and implementing effective retention strategies.



Enhance Customer Satisfaction

Leverage predictive insights to proactively address customer needs and improve overall satisfaction.



Drive Business Growth Growth

Provide telecom clients with a competitive advantage through our advanced churn prediction capabilities.



Optimize Business Performance

Advanced analytics and machine learning are used in telecom churn prediction to identify customer churn factors, enabling proactive retention strategies to reduce attrition and boost profitability.

key challenges



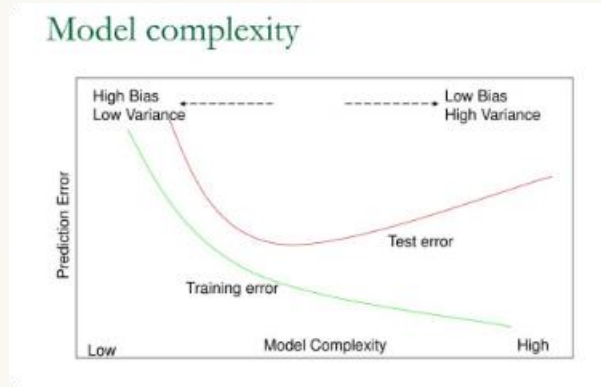
Data Quality and Completeness

Telecom data can be noisy, incomplete, and distributed across multiple systems. Ensuring high-quality, consistent data is critical for model performance.



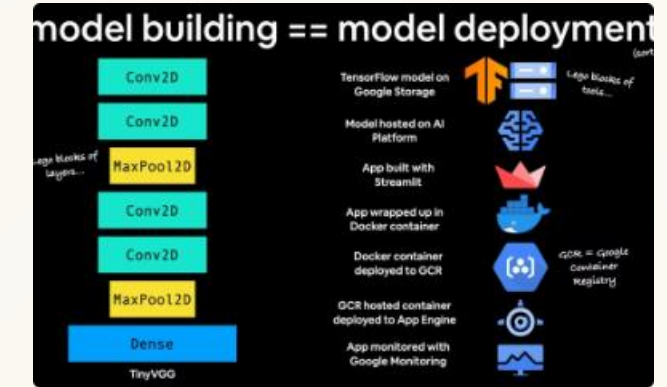
Feature Engineering

Identifying the right set of customer attributes and behavioral signals that influence churn is crucial. This requires deep domain expertise and extensive experimentation.



Model Complexity

Churn is a complex phenomenon influenced by a variety of factors. Developing a model that can capture these nuances while maintaining interpretability is a delicate balance.



Deployment and Scalability

The final model must be deployable at scale, with the ability to continuously ingest new data and generate real-time predictions to enable timely interventions.

Data Cleaning and Preprocessing

1

Data Exploration

Understand the structure, characteristics, and potential issues in the customer data.

2

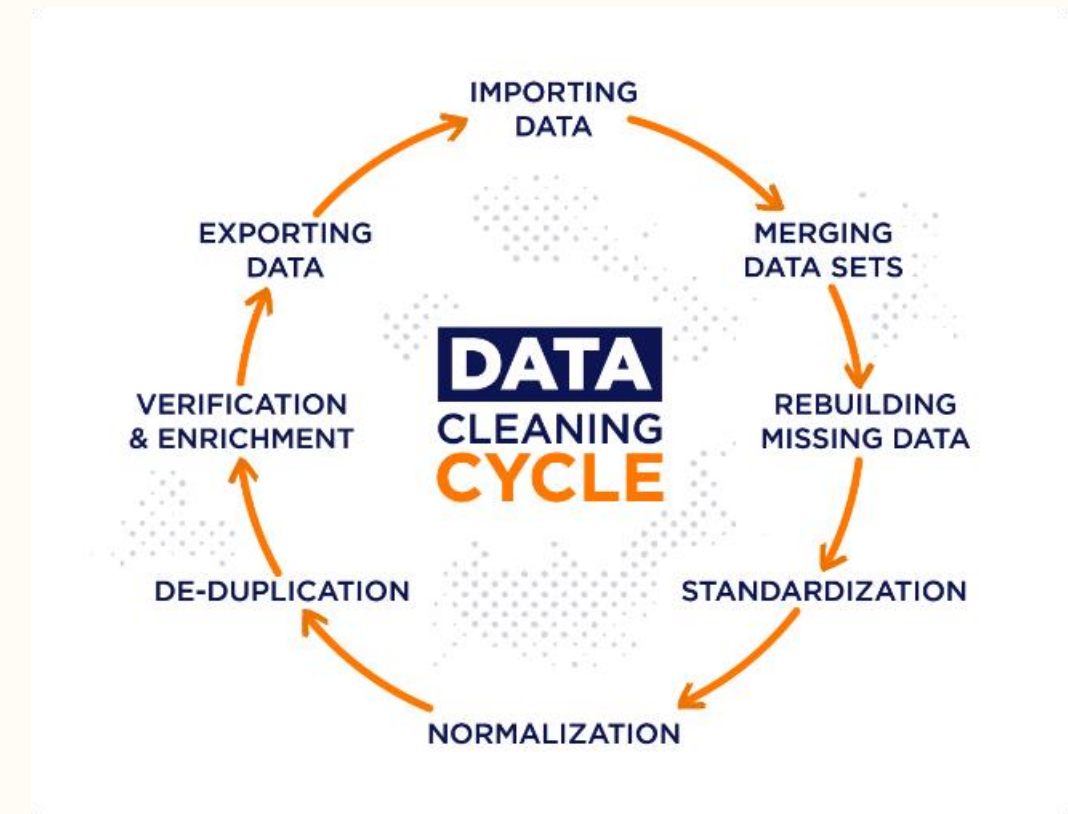
Data Cleaning

Address missing values, outliers, and inconsistencies to ensure data integrity.

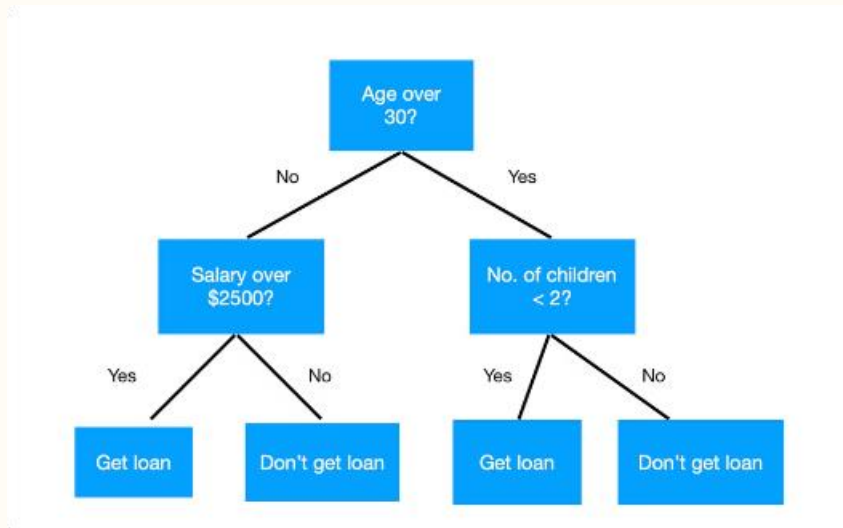
3

Feature Engineering

Create new, meaningful features from the raw data to enhance the predictive power.

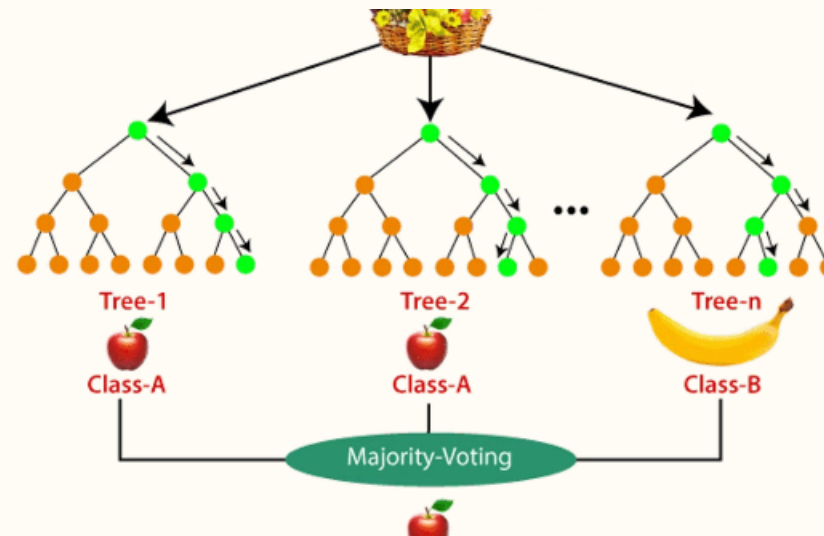


Machine Learning Model Building



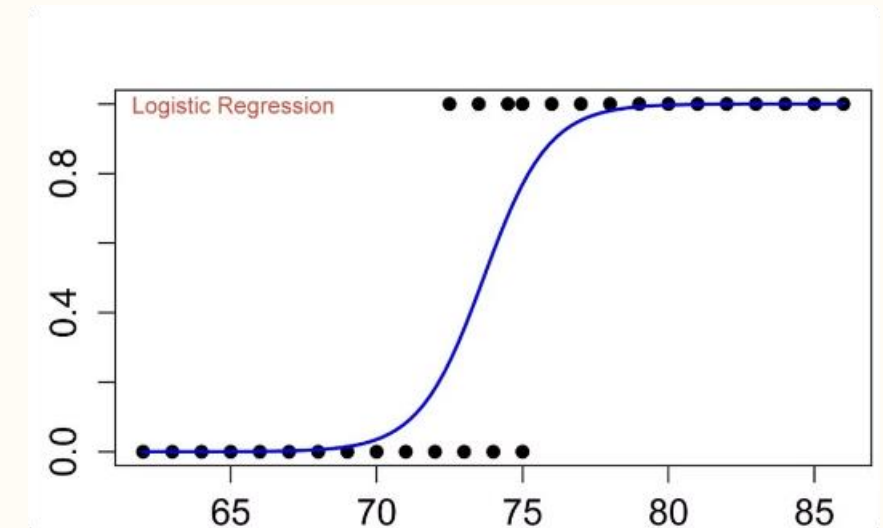
Decision Trees

Powerful classifiers that can capture non-linear relationships in the data.



Random Forest

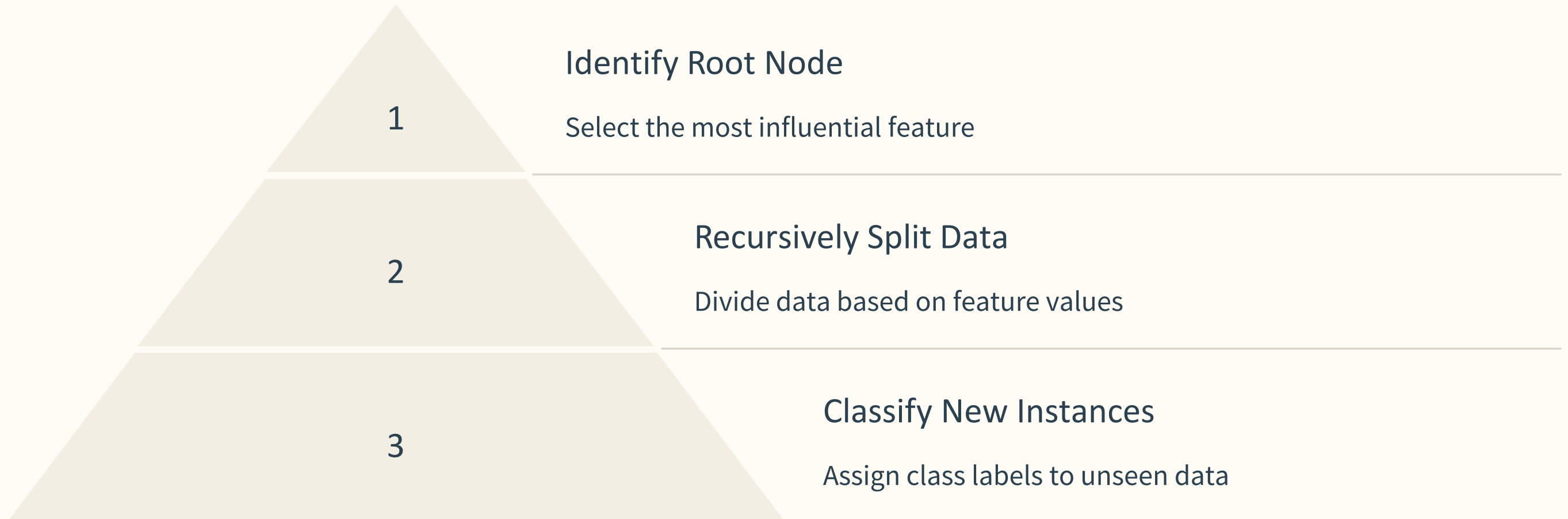
Ensemble methods that combine multiple decision trees for improved accuracy.



Logistic Regression

Widely used for binary classification tasks, such as predicting customer churn.

Decision Tree



Decision trees are a powerful machine learning algorithm that can capture complex, non-linear relationships in the data. They work by recursively partitioning the dataset based on the most informative features, creating a tree-like structure of decisions that can be used to make predictions on new, unseen data.

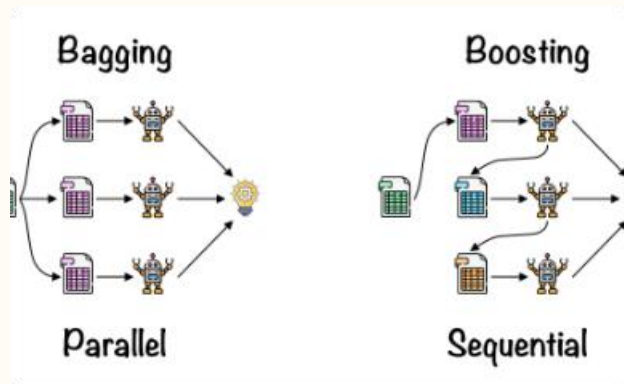
Code Snippet

```
# Selecting Features
features = churn_data.columns.drop("target")
target = ["target"]
# Encoding categorical variables
churn_data_encoded = pd.get_dummies(churn_data[features])
churn_data_encoded["target"] = churn_data["target"]
# Splitting the data into train and test set in a ratio of 85:15
churn_train, churn_test = train_test_split(churn_data_encoded, test_size=0.15, random_state=100)
# Inducting the decision tree model
model = DecisionTreeClassifier()
features = churn_train.columns.drop("target")
model.fit(churn_train[features], churn_train[target])
```

Test set accuracy with Randomized Search: 0.7765333333333333

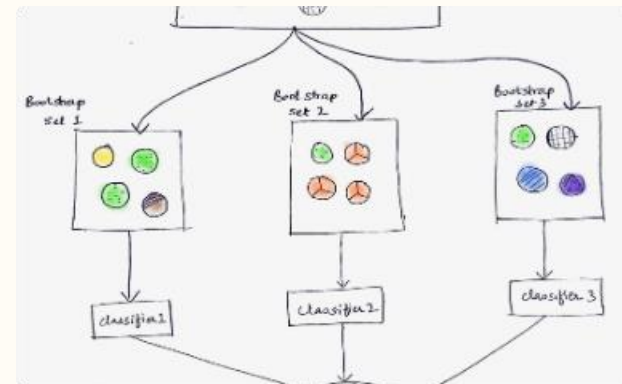
Confusion Matrix:
[[2241 312]
[526 671]]
Accuracy: 0.7765333333333333
Precision: 0.6826042726347915
Recall: 0.5605680868838764
F1-Score: 0.6155963302752294

Random Forest



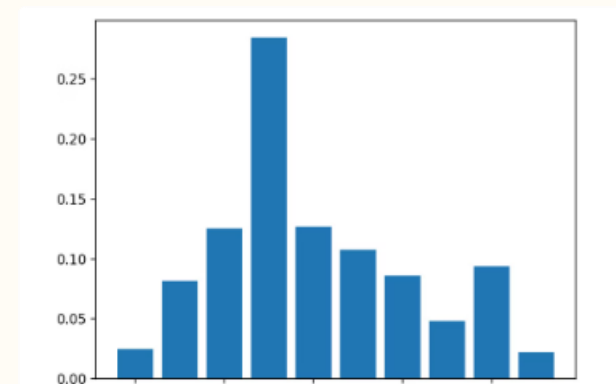
Ensemble Method

Random Forest combines multiple decision trees to improve accuracy and robustness.



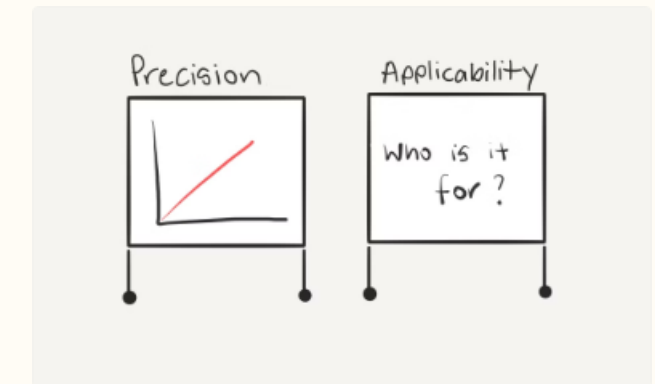
Bagging Technique

The algorithm trains each tree on a random subset of the data, allowing the model to capture different patterns and reduce overfitting.



Feature Importance Importance

Random Forest can provide insights into the relative importance of different features, guiding feature engineering and selection.



Versatile Applicability Applicability

This algorithm can handle a wide range of data types, including numerical, categorical, and mixed features.

Code Snippet

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler

# Reading the churn data from a CSV file
churn_data = pd.read_csv("C:\\Users\\Divya Bagade\\OneDrive\\Desktop\\Churn_ Data.csv")

# Selecting Features and Target
features = churn_data.columns.drop("target")
target = "target"

# Encoding categorical variables
churn_data_encoded = pd.get_dummies(churn_data[features], drop_first=True)
churn_data_encoded[target] = churn_data[target]

# Splitting the data into train and test set in a ratio of 85:15
churn_train, churn_test = train_test_split(churn_data_encoded, test_size=0.15, random_state=100)

# Feature scaling (optional, but can improve model performance)
scaler = StandardScaler()
churn_train[features] = scaler.fit_transform(churn_train[features])
churn_test[features] = scaler.transform(churn_test[features])
```

Test set accuracy: 0.79

```
# Inducting the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(churn_train[features], churn_train[target])

# Evaluate the Model
accuracy = rf_model.score(churn_test[features], churn_test[target])
print(f"Test set accuracy: {accuracy:.2f}")

# Predict probabilities for the test set
y_pred_prob = rf_model.predict_proba(churn_test[features])[:, 1]
```

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.89	0.85	2553
1	0.71	0.58	0.64	1197
accuracy			0.79	3750
macro avg	0.77	0.74	0.75	3750
weighted avg	0.79	0.79	0.79	3750

Logistic Regression

1

Probabilistic Approach

Logistic Regression outputs the probability of an instance belonging to a particular class, rather than a hard classification.

2

Flexibility

The algorithm can handle a variety of data types, including numerical, categorical, and mixed features.

3

Interpretability

Logistic Regression models provide insights into the relative importance of different features, making it easier to understand the driving factors behind the predictions.



```

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, roc_auc_score, confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler

# Reading the churn data from a CSV file
churn_data = pd.read_csv("C:\\Users\\Divya Bagade\\OneDrive\\Desktop\\Churn_ Data.csv")

# Selecting Features and Target
features = churn_data.columns.drop("target")
target = "target"

# Encoding categorical variables
churn_data_encoded = pd.get_dummies(churn_data[features], drop_first=True)
churn_data_encoded[target] = churn_data[target]

# Splitting the data into train and test set in a ratio of 85:15
churn_train, churn_test = train_test_split(churn_data_encoded, test_size=0.15, random_state=100)

# Feature scaling (optional, but can improve Logistic regression performance)
scaler = StandardScaler()
churn_train[features] = scaler.fit_transform(churn_train[features])
churn_test[features] = scaler.transform(churn_test[features])

```

```

Test set accuracy: 0.7941333333333334

```

```

# Inducting the Logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(churn_train[features], churn_train[target])

# Evaluate the Model
accuracy = model.score(churn_test[features], churn_test[target])
print(f"Test set accuracy: {accuracy}")

# Predict probabilities for the test set
y_pred_prob = model.predict_proba(churn_test[features])[:, 1]

```

Classification Report:				
	precision	recall	f1-score	support
0	0.82	0.89	0.85	2553
1	0.72	0.59	0.65	1197
accuracy			0.79	3750
macro avg	0.77	0.74	0.75	3750
weighted avg	0.79	0.79	0.79	3750

Code Snippet

Key Difference

Parameters	Decision Tree	Logistic Regression	Random Forest
Training Accuracy	0.77	0.79	0.79
Testing Accuracy	0.77	0.79	0.79
Precision	0.68	(0) 0.82 (1) 0.72	(0) 0.82 (1) 0.71
Recall	0.56	(0) 0.89 (1) 0.59	(0) 0.89 (1) 0.58

Model Evaluation and Validation

1

Cross-Validation

Assess model performance on multiple training and testing splits to ensure robustness.

2

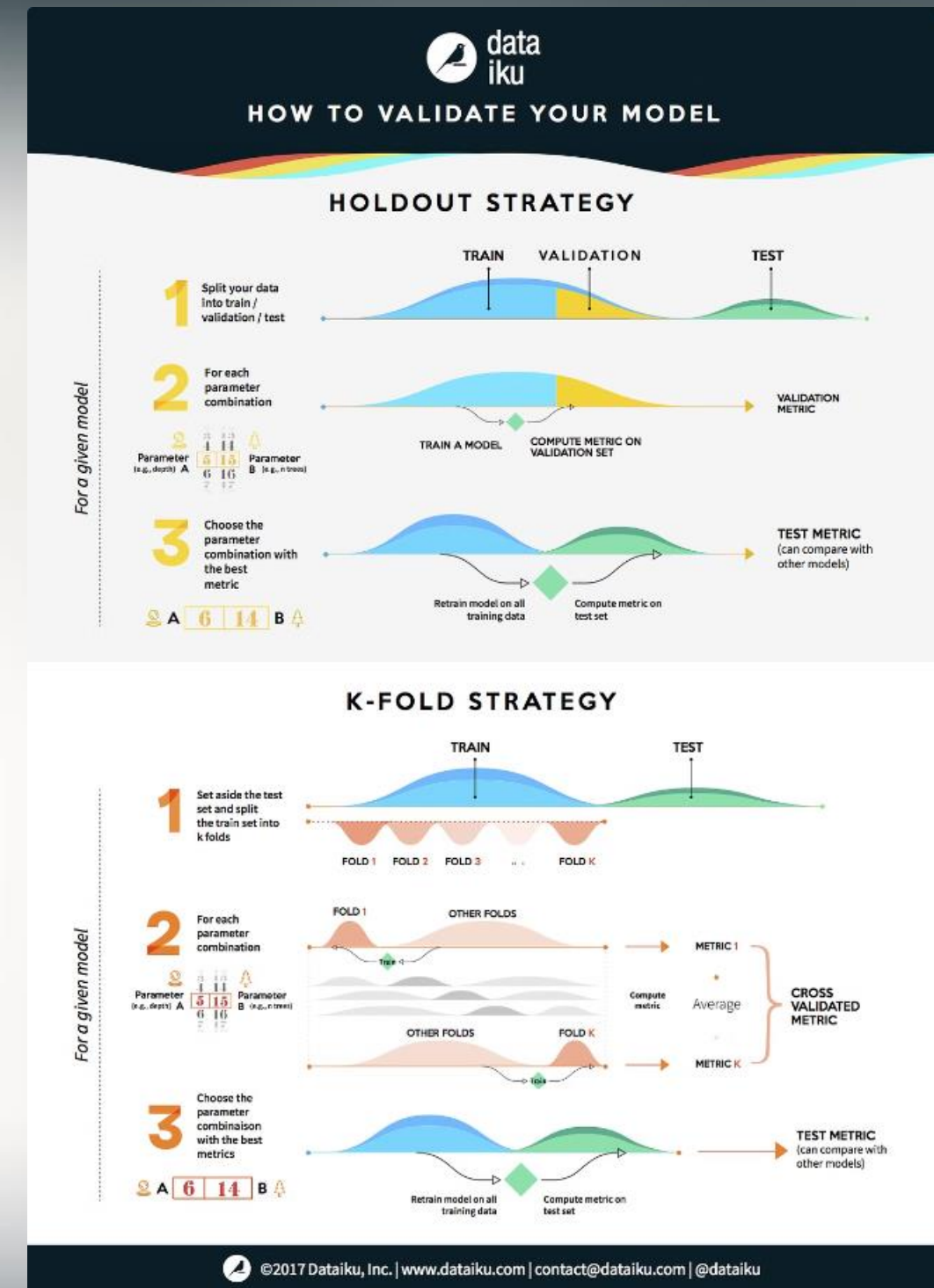
Metrics Evaluation

Measure the model's accuracy, precision, recall, and F1-score to determine its effectiveness.

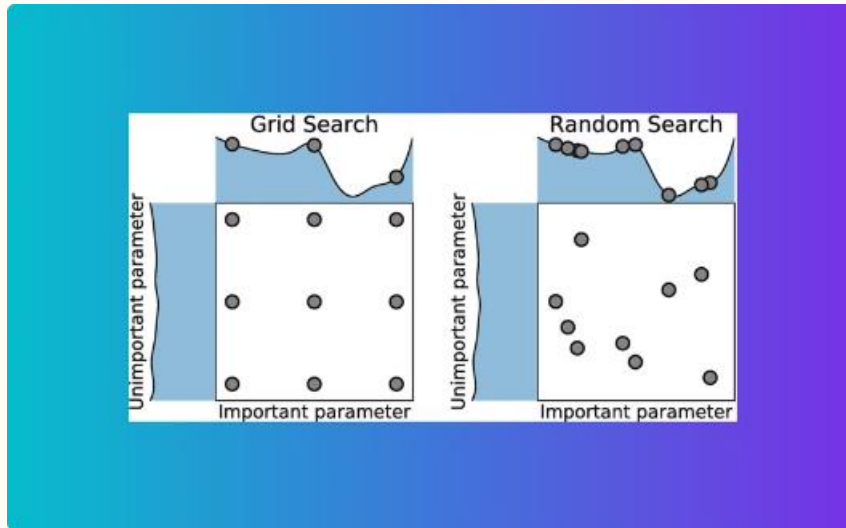
3

Holdout Testing

Evaluate the model's generalization capability on an unseen, independent dataset.

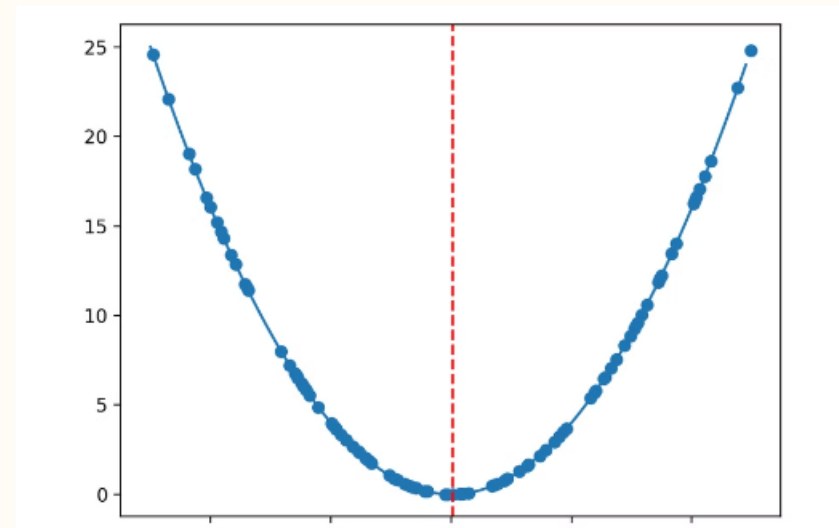


Hyperparameter Tuning



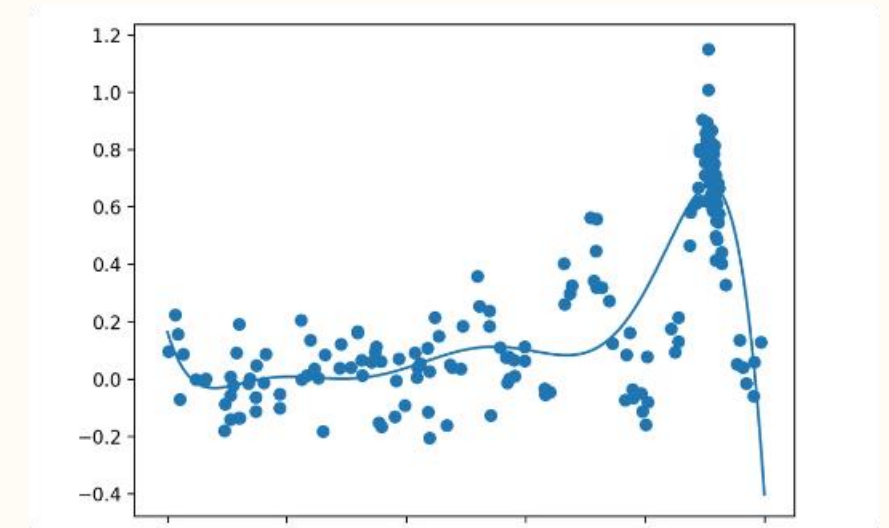
Grid Search

Systematically explore a predefined set of hyperparameters to find the optimal configuration.



Random Search

Randomly sample hyperparameter values within a specified range to identify the best model.



Bayesian Optimization

Use a probabilistic model to intelligently navigate the hyperparameter space and converge on the optimal settings.


```
# Hyperparameter Tuning with Randomized Search
```

```
param_dist = {  
    'criterion': ['gini', 'entropy'],  
    'splitter': ['best', 'random'],  
    'max_depth': [None, 10, 20, 30, 40, 50],  
    'min_samples_split': randint(2, 21),  
    'min_samples_leaf': randint(1, 11)  
}  
  
random_search = RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=100, cv=5, scoring='accuracy',  
                                   n_jobs=-1, random_state=42)  
random_search.fit(churn_train[features], churn_train[target].values.ravel())  
print(f"Best parameters found (Randomized Search): {random_search.best_params_}")  
print(f"Best accuracy score (Randomized Search): {random_search.best_score_}")
```

```
Best parameters found (Randomized Search): {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 5, 'min_samples_split': 10, 'splitter': 'random'}
```

```
Best accuracy score (Randomized Search): 0.7760941176470588
```

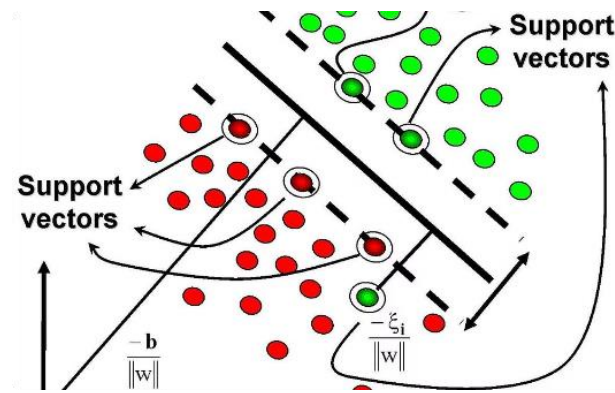
```
# Evaluate the Best Model
```

```
best_model_random = random_search.best_estimator_  
accuracy_random = best_model_random.score(churn_test[features], churn_test[target])  
print(f"Test set accuracy with Randomized Search: {accuracy_random}")
```

```
Test set accuracy with Randomized Search: 0.7765333333333333
```

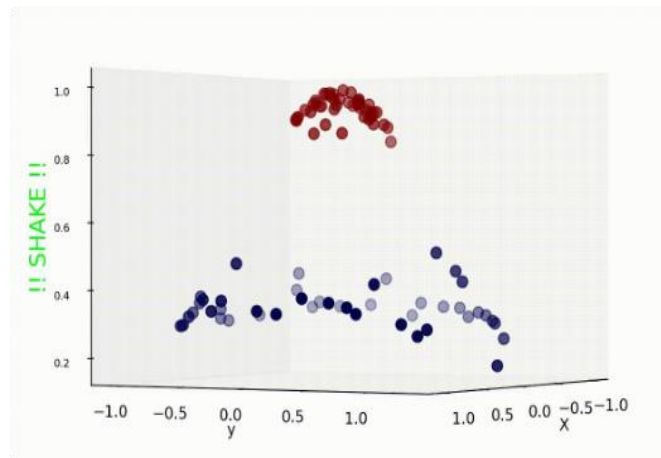
Code Snippet

Support Vector Machine Machine



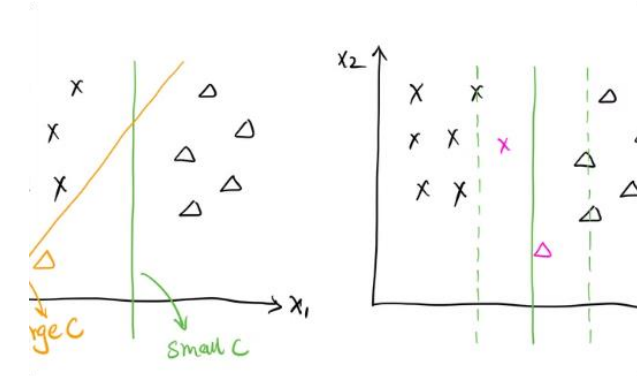
Geometric Approach

Support Vector Machines (SVMs) use a geometric approach to find the optimal hyperplane that separates different classes with the maximum margin.



Kernel Trick

The kernel trick allows SVMs to handle non-linear decision boundaries by mapping the data to a higher dimensional space where it becomes linearly separable.



Regularization

Regularization techniques like C-parameter and gamma help SVMs balance model complexity and generalization, preventing overfitting.

Code Snippet

```
Classification Report:
              precision    recall  f1-score   support

     0           0.82       0.89       0.85        2553
     1           0.72       0.59       0.65        1197

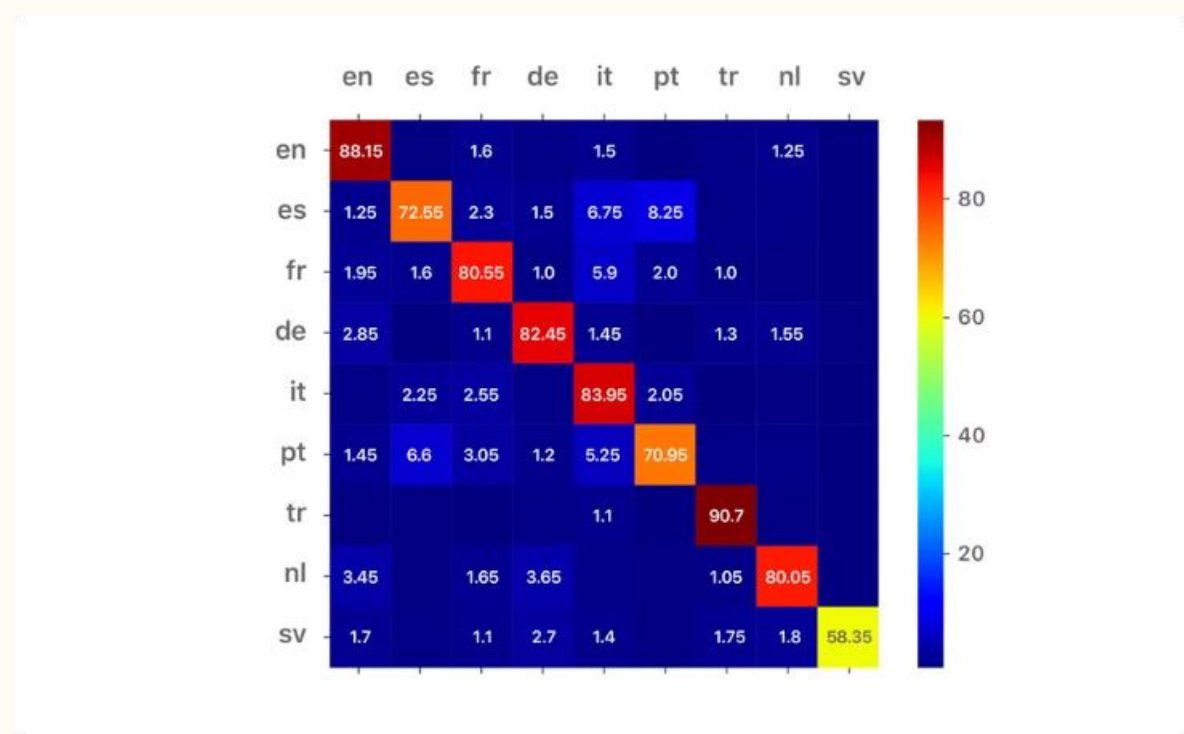
 accuracy          0.79        3750
 macro avg         0.77        3750
weighted avg         0.79        3750
```

```
Classification Report:
              precision    recall  f1-score   support

     0           0.82       0.89       0.85        2553
     1           0.71       0.58       0.64        1197

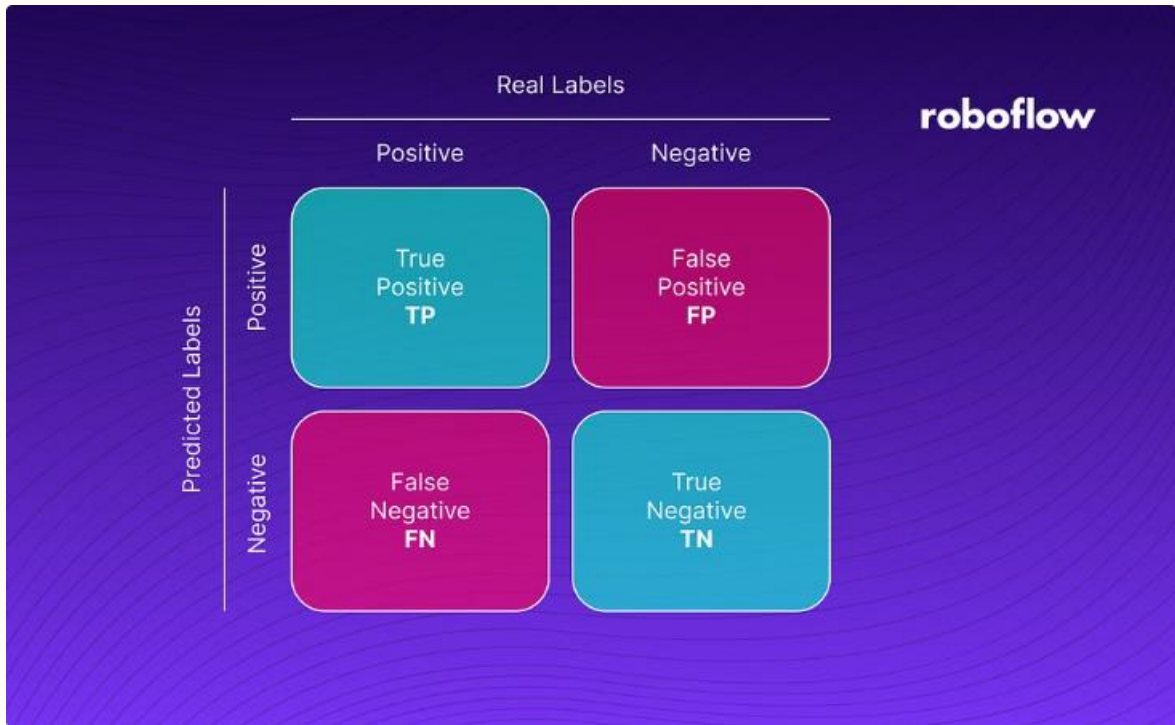
 accuracy          0.79        3750
 macro avg         0.77        3750
weighted avg         0.79        3750
```


Leveraging the Power of the Confusion Matrix Matrix



Confusion Matrix

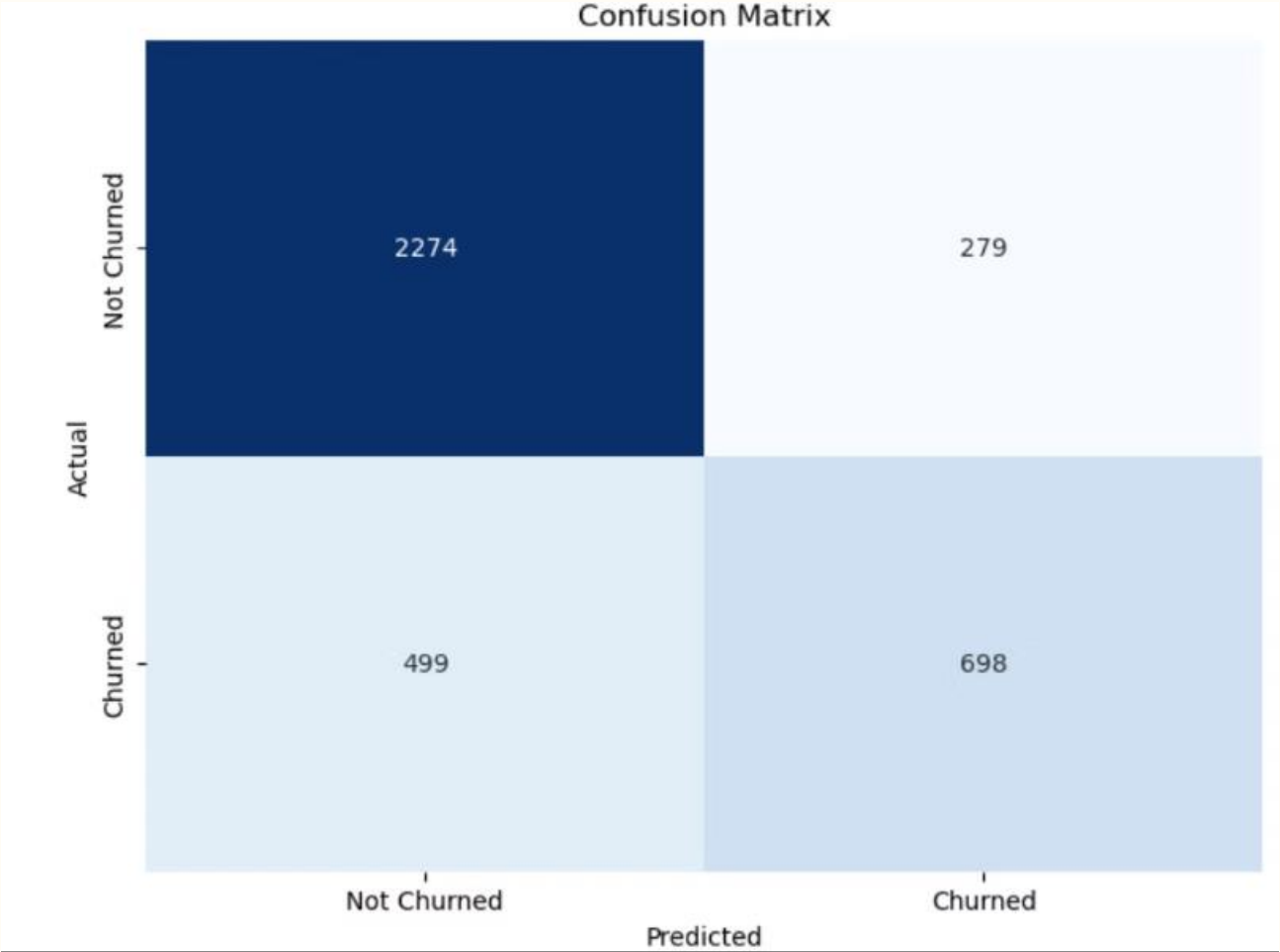
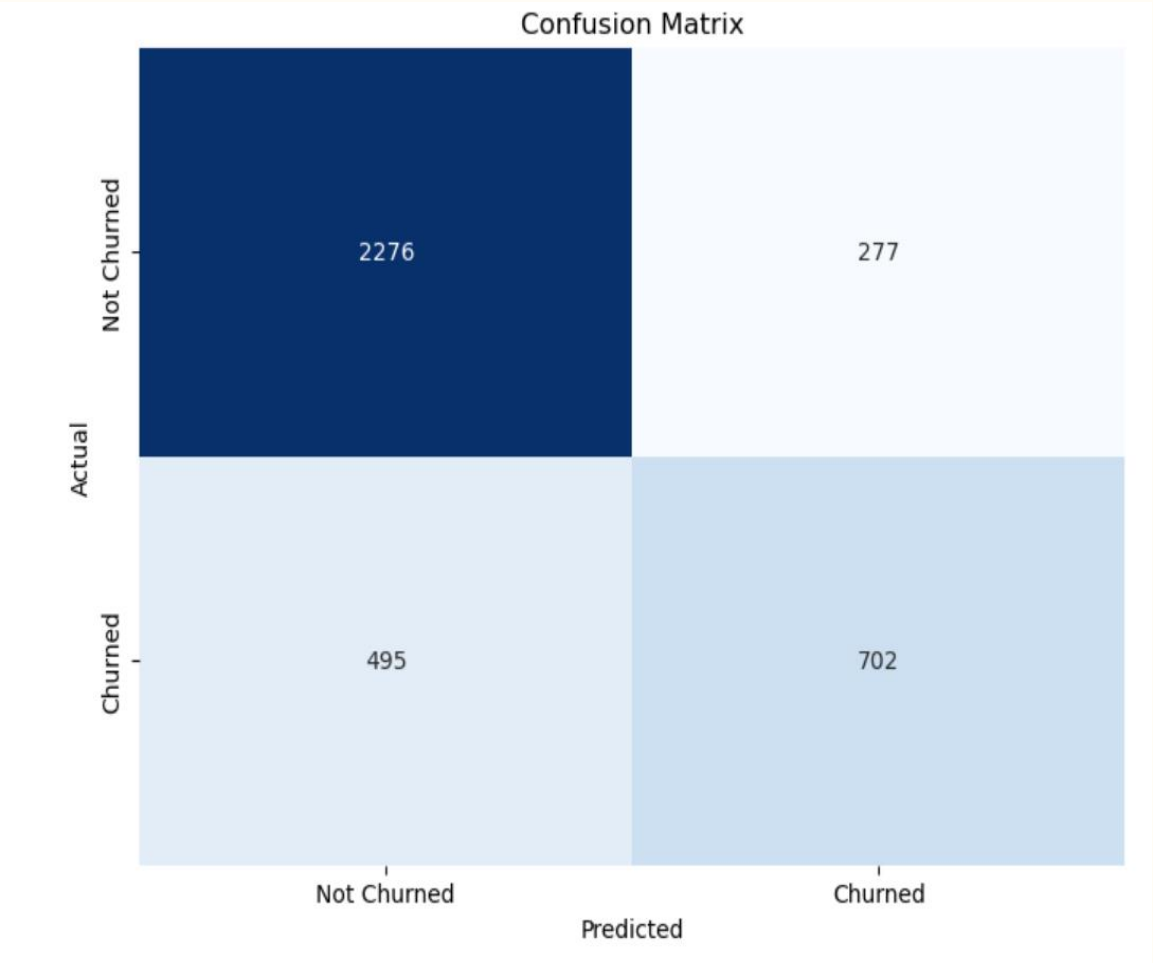
A confusion matrix is a visual tool that provides a comprehensive evaluation of the model's predictive performance, including accuracy, precision, recall, and F1-score.



Insights from Confusion Matrix

The confusion matrix helps identify areas for improvement, such as false positives or false negatives, to enhance the overall model performance.

Code snippet



Conclusion and Future Considerations

1 Effective Churn Prediction

The developed model demonstrates a robust and accurate approach to predicting customer churn in the telecom industry.

2 Ongoing Optimization

Continuous monitoring, model updates, and incorporation of new data sources will ensure the sustained accuracy and relevance of the churn prediction solution.

3 Expanding Capabilities

Exploring advanced techniques like deep learning and leveraging external data sources can further enhance the predictive capabilities and uncover deeper insights.

