rurigi-waweru / dsfpt10-p3-end-of-phase-3-project

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

dsfpt10-p3-end-of-phase-3-project

Public

1 Branch 0 Tags Go to file Add file

rurigi-waweru deliverables-v1.0 -b ea2426d · 1 minute ago 35 Commits

images	added-working-images -b	9 minutes ago
others	remastered-files	4 minutes ago
working-text-files	enveloped-working-files -b	yesterday
README.md	updated-presentation-power...	10 minutes ago
churn_in_telecoms_dat...	added-csv-file -b	yesterday
notebook.ipynb	updated-submission-READM...	1 hour ago
notebook.pdf	deliverables-v1.0 -b	1 minute ago
presentation.pdf	updated-presentation-power...	10 minutes ago
presentation.pptx	updated-presentation-power...	10 minutes ago

About

End of Phase 3 Project: My first classification project, where I built and tuned machine learning classification models to answer stakeholder questions and provide additional insights.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

README

Contributors: Brian Waweru, Start-Date : 08th May, 2025

Languages

● **Jupyter Notebook** 100.0%

1.1.0 Overview

The project aims to predict customer churn for SyriaTel, a telecommunications company, using historical customer data. The core goal is to build a robust machine learning classification model that can flag at-risk customers, enabling the company to implement effective retention strategies and reduce revenue loss.

1.1.1: Working Libraries and Preliminaries

First, we set up the necessary tools for a machine learning classification workflow in Python. The various libraries are imported to support at a glance were Data handling (pandas), Data preprocessing (StandardScaler, OneHotEncoder, ColumnTransformer), Model building (Logistic Regression, Random Forest, XGBoost, SVC, KNN), Pipeline creation and model evaluation (Pipeline, accuracy_score, ROC metrics, confusion matrix), Addressing class imbalance (SMOTE), Label encoding (LabelEncoder), Plotting and visualization (matplotlib.pyplot)

This selection of models and tools highlight that the code will perform binary or multiclass classification, an imbalanced dataset, with preprocessing steps included in a pipeline.

1.1.2: Basic Exploratory Data Analysis

After importing the file into the python notebook. The dataset was seen to have 3333 rows and 21 columns i.e. features. One of the column was `churn` . Since this is a churn dataset, the target variable was certainly `churn` column with the rest being the other `features` .

Afterwards, the eda revealed that some few columns were categorical in nature with `object` variables. The `churn` column was streamlined to be `binary` while the reset we either dropped (i.e. `phone number`) or converted to `binary` as well.

`No` row had a missing entry! The `area code` had only 3 unique entries

2.1.0: Overview-Introduction: Customer Churn Prediction for SyriaTel

This project aims to predict customer churn for `SyriaTel` , a telecommunications company, using a sample of their historical customer data. By building a binary classification model since the customer either churns '1' or does not '0', we shall aim to identify patterns and factors that influence whether a customer will leave the company. The predictive model will assist the company in targeting at-risk customers with `retention strategies` , thereby reducing customer attrition and preserving revenue.

The overall project pipeline consists of:

1. **Business Understanding**: Understanding churn's impact on SyriaTel’s business.
2. **Data Understanding and Preparation**: Exploring the structure and distribution of data. Cleaning, transforming, and encoding the dataset.
3. **Exploratory Data Analysis (EDA)**: Finding patterns and feature relationships with churn.
4. **Model Building**: Training and tuning classifiers such as Logistic Regression, Decision Trees or Random Forests.
5. **Evaluation**: Measuring performance using metrics like accuracy, precision, recall, F1-score, and AUC as well as ROC.
6. **Interpretation**: Identifying key drivers of churn.
7. **Recommendations and Actionable Insights**: Informing business interventions to reduce churn. Provide recommendations for customer retention based on analytical findings.

2.1.1: Project-Objectives

Here are the Objectives in this project:-

1. **Build a Predictive Model for Churn**
2. **Improve Churn Prediction Accuracy**
3. **Develop a Repeatable ML Pipeline**: Build a clean and modular workflow that can be reused with updated customer data in the future.
4. **Then Communicate Findings Clearly**: Present model insights

3.1.0 : Business and Data Understanding

3.1.1: Business Understanding

Customer churn is a critical business challenge for telco companies such as SyriaTel. In a highly competitive and saturated market, retaining existing customers is often more cost-effective than acquiring new ones. Churn not only impacts immediate revenue but also affects long-term customer lifetime value, brand loyalty, and operational efficiency. Understanding why customers leave — and more importantly, identifying who is likely to leave — can empower SyriaTel to take timely, targeted actions. These may include `personalized marketing campaigns`, `service improvements`, or `tailored retention offers`.

The core business goal of this project is to reduce churn by building a predictive model that accurately flags at-risk customers. This enables SyriaTel to shift from reactive to proactive customer retention, thereby reducing revenue loss and enhancing customer satisfaction.

The project aligns with SyriaTel's strategic priorities:

1. `Preserving revenue` by minimizing customer loss.
2. `Improving customer loyalty` through better engagement.
3. `Increasing the return on investment (ROI)` of marketing and support efforts.
4. `Leveraging data` to drive smarter, faster business decisions.

Ultimately, this project supports SyriaTel’s mission to build lasting customer relationships in a competitive telecom landscape.

3.1.2: EDA: Data Understanding

The dataset provided by SyriaTel consists of over 3300 customer records and 21 features, each capturing various aspects of a customer's interaction with the company's service. The target variable is `churn`, which indicates whether a customer has discontinued service or not. Understanding the composition and behavior of these column-features is critical in helping us build an effective churn prediction model.

1. Several features describe customer demographics and account information, such as `state`, `area code`, and `account length`. While these may not directly cause churn, they can help identify regional trends or the effect of customer tenure on loyalty.
2. Other features capture service plans (`international plan`, `voice mail plan`), indicate whether a customer is subscribed to specific services. These features may influence customer satisfaction and costs, potentially affecting their decision to stay or leave.
3. A significant portion of the dataset focuses on usage behavior, including `call minutes`, `number of calls`, and `charges` during the day, evening, night, and for international calls. These metrics are split into separate fields for minutes, calls, and charges. This could allow an examinantion of customer engagement and how it relates to churn. However, since charges are typically derived from minutes, some of these columns may be redundant.
4. The dataset also includes features such as the number of `customer service calls`, which can be a strong indicator of dissatisfaction—customers who contact support frequently may be more likely to churn.
5. Importantly, the dataset is clean, with `no missing values`, and the data types are appropriate for analysis—numerical for continuous variables and object or boolean for categorical ones. However, some preprocessing will be necessary, including encoding categorical variables and dropping non-informative columns like phone number, as done previously, which acts only as an identifier.

Through a Thorough EDA, we aim to understand the relationships between these features and the likelihood of churn. Identifying patterns, such as whether certain service plans correlate with higher churn, or whether customers with higher international usage are more likely to leave, will help us build a predictive model and generate actionable business insights.

4.1.0 : Data Preparation

4.1.1: Handle Categorical Variables

```
# Categories i.e. classify the values in the 'international plan' as either 1 or 0
df['international plan'] = df['international plan'].map({'yes': 1, 'no': 0})
# Categories i.e. classify the values in the 'voice mail plan' as either 1 or 0
df['voice mail plan'] = df['voice mail plan'].map({'yes': 1, 'no': 0})
```

4.1.2 : Drop Irrelevant and Redundant Columns

As mentioned earlier in the overview and business understanding, features like `total day charge` might be redundant if `total day minutes` already provides similar information. You might choose to drop one.

- `note:` The feature `phone number` had been dropped already. This is because it only serves as a customer identifier.
- `note:` As noted earlier, there are no missing values. See section under `df.info()` .

4.1.3 : Redundant Columns

```
redundant_columns = ['total day charge', 'total eve charge', 'total night charge', 'total intl charge']
# dropping
df.drop(redundant_columns, axis=1, inplace=True)
```

4.1.4 : Standardization

Now, we transform features in the dataset i.e. `total day minutes` , `number vmail messages` , `total eve minutes` so that they have a common scale.

```
# Create a scaler object
scaler = StandardScaler()

# Choice of columns to standadise
cols_to_standardize = ['total day minutes', 'number vmail messages', 'total eve minutes']

# Apply standardization to relevant columns
df[cols_to_standardize] = scaler.fit_transform(df[cols_to_standardize])
```

4.1.5 : Feature Engineering

- Since the dataset is riddled with `minutes` , suppose we have `Total Call Usage` i.e the sum of `total day minutes` , `total eve minutes` , `total night minutes` , and `total intl minutes` .

- Also, we can have `Average Call Duration` i.e. for Average of `day` , `evening` , `night` , and `international minutes` .

```
# TOTAL CALL USAGE:
df['total minutes'] = df['total day minutes'] + df['total eve minutes'] + df['total night minutes'] + df['total intl minutes']
# AVERAGE CALL DURATION:
df['average call duration'] = df[['total day minutes', 'total eve minutes', 'total night minutes', 'total intl minutes']].mean(
```

4.1.6 : Choosing `Target` and `Feature` column(s)

It is obvious that the choice of our Target column is `churn` while the rest are automatically the `Features` .

Now, Splitting the Data into `Training` and `Test` dataSets

```
# Target Feature ## Dependent Feature
y = df.churn
# Other Features ## independent Features
X = df.drop('churn', axis=1)
# split-test-code
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

4.1.7 : Check for Class Imbalance

Let's ensure that the model doesn't learn misleading patterns — especially because we have binary classification problem.

```
# first, let's check class distribution
print(f"The Value counts are: \n{df.churn.value_counts()}", end = '\n\n')
# The proportions are:
print('Essentially, that is:-', end = '\n')
print(f"{df['churn'].value_counts(normalize=True)}", end = '\n')
# Inference and Conclusion
churn_perc = round(df['churn'].value_counts(normalize=True)[1] * 100, 2)
# print the result
```

```
print(' ')
print(f"INFERENCE: So, only {churn_perc}% of the customers churn — this shows class imbalance.")
```

Inference:

This class imbalance refers to the fact that one class (non-churn) significantly outweighs the other `churning` group. This imbalance can affect the performance of machine learning model. They may become biased toward predicting the majority class, the `non-churning`, which could result in misleading accuracy scores.

```
# RE_DONE
X_encoded = pd.get_dummies(X, drop_first=True)  # drop_first avoids dummy trap

#
le = LabelEncoder()
X['state'] = le.fit_transform(X['state'])

# Encode categorical variables
X_encoded = pd.get_dummies(X, drop_first=True)
# Step 1: Encode categorical variables (e.g., 'State', 'Gender', etc.)
X_encoded = pd.get_dummies(X, drop_first=True)  # One-Hot Encoding

# Step 2: Split data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X_encoded, y, stratify=y, test_size=0.2, random_state=42)

# Step 3: Apply SMOTE to oversample the minority class in the training data
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Step 4: Train a Random Forest Classifier with class weights to handle class imbalance
clf = RandomForestClassifier(class_weight='balanced', random_state=42)
clf.fit(X_train_res, y_train_res)

# Step 5: Make predictions on the test set
y_pred = clf.predict(X_test)
y_proba = clf.predict_proba(X_test)[:, 1]
```

```
# Step 6: Evaluate the model performance
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("ROC-AUC Score:", roc_auc_score(y_test, y_proba))
```

5.1.0: Modeling and Evaluation

5.1.1 : Logistic Regression

Let us start with a `simple Logistic Regression` model, before we try others like Random Forest. model = LogisticRegression(max_iter=1000, class_weight='balanced') # Use class_weight if you didn't use SMOTE model.fit(X_train, y_train)

5.1.2 : Others

```
# One-hot encode categorical variables
X = pd.get_dummies(X, drop_first=True)

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42)

# Feature Scaling (optional, but needed for SVM, KNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

5.1.3 : Evaluation

Train and evaluate each model.

```
# Define models
models = {
    "Logistic Regression": LogisticRegression(class_weight='balanced', max_iter=10000),
    "Random Forest": RandomForestClassifier(class_weight='balanced', random_state=42),
    # "XGBoost": XGBClassifier(scale_pos_weight=6, use_label_encoder=False, eval_metric='logloss', random_state=42),
    # "SVM": SVC(class_weight='balanced', probability=True, random_state=42),
    "KNN": KNeighborsClassifier()
}

# Train and evaluate each model

for name, model in models.items():

    print(f"\n----- {name} -----")

    if name in ["SVM", "KNN"]:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        y_proba = model.predict_proba(X_test_scaled)[:, 1]
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_proba = model.predict_proba(X_test)[:, 1]

    print(classification_report(y_test, y_pred))
    print("ROC AUC:", roc_auc_score(y_test, y_proba))
```

5.1.4 : Logistical Regression

After training the model, it's important to evaluate its performance using:

1. `Accuracy` : The percentage of correct predictions.
2. `Confusion Matrix` :
   - Helps you understand the model’s performance with respect to false positives, false negatives, true positives, and true negatives.

3. `Precision, Recall, F1-Score` :
   - Especially important for imbalanced datasets or when the costs of false positives/negatives differ.
4. `ROC-AUC Curve` : Evaluate the classifier's ability to distinguish between classes.

The code:-

```
# Train the Model
# Fit a Logistic Regression model
Model = LogisticRegression()
Model.fit(X_train_scaled, y_train)

# Get prediction Probabilities
# Predict probability estimates for the positive class
y_probs = Model.predict_proba(X_test_scaled)[:, 1]

# Computing ROC Curve and AUC
# Compute False Positive Rate and True Positive Rate
fpr, tpr, thresholds = roc_curve(y_test, y_probs)

# Calculate the AUC score
auc = roc_auc_score(y_test, y_probs)
```

5.1.6 : Plot the ROC Curve

Plotting the ROC Curve:-

```
# Plot the ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Random Classifier')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve',
            fontweight = 'bold', fontsize = 16)
plt.legend(loc = 4)
plt.grid(True)
plt.show()
```

Evaluating:-

```
### Evaluate
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Initialize a plot
plt.figure(figsize=(10, 8))

# Train and evaluate each model
for name, model in models.items():
    print(f"\n----- {name} -----")

    # Use scaled or unscaled data depending on the model
    if name in ["SVM", "KNN"]:
        model.fit(X_train_scaled, y_train)
        y_pred = model.predict(X_test_scaled)
        y_proba = model.predict_proba(X_test_scaled)[:, 1]
    else:
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
        y_proba = model.predict_proba(X_test)[:, 1]

    # ROC curve values
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, label=f'{name} (AUC = {roc_auc:.2f})')

    # Optionally print performance
    print(classification_report(y_test, y_pred))
    print("ROC AUC:", roc_auc)

# Plot reference line
plt.plot([0, 1], [0, 1], 'k--', label='Random Guess')

# Plot settings
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

6.1.0 : Conclusions and Recommendations

The ROC curve showing how your classifier performs across different thresholds. The AUC value, 0.90 summarizing overall performance for the