**PHASE 5 PROJECT SUBMISSION**

**PROJECT 6 - CUSTOMER CHURN PREDICTION**

**TEAM MEMBERS:**

1. Rithees S M - 2021504032
2. Akshay G S – 2021504502
3. Dhanesh C N E – 2021504510
4. Divyavarshini K – 2021504513

**Problem Definition:**

The project involves using IBM Cognos to predict customer churn and identify factors influencing customer retention. The goal is to help businesses reduce customer attrition by understanding the patterns and reasons behind customers leaving. This project includes defining analysis objectives, collecting customer data, designing relevant visualizations in IBM Cognos, and building a predictive model.

**Phase Objective:**

Outline the project's objective, design thinking process, and development phases. Describe the analysis objectives, data collection process, data visualization using IBM Cognos, and Python code integration. Explain how the insights from the analysis can help website owners improve user experience.

**Dataset Link:**

[**https://www.kaggle.com/datasets/blastchar/telco-customer-churn**](https://www.kaggle.com/datasets/blastchar/telco-customer-churn)

**Synopsis:**

In this phase we reduce the class imbalance in the dataset using SMOTE analysis .Used RFE for feature selection. Carried out Model ensembling using a voting classifier with XGBoost and Random Forest. Adjusted the threshold for classification based on the F1-score to balance precision and recall. Included the reg\_lambda hyperparameter which is L2 regularization term on weights. This, in combination with previous enhancements, should further improve the model's performance.

**SMOTE Analysis :**

print("Distribution of target variable in training set before applying SMOTE: ", y\_train.value\_counts(), sep='\n')

sm = SMOTE(random\_state=123)

X\_train\_sm, y\_train\_sm = sm.fit\_resample(X\_train, y\_train)

print("\nDistribution of target variable in training set after applying SMOTE: ", y\_train\_sm.value\_counts(), sep='\n')

print("Distribution of target variable in testing set before applying SMOTE: ", y\_test.value\_counts(), sep='\n')

sm = SMOTE(random\_state=123)

X\_test\_sm, y\_test\_sm = sm.fit\_resample(X\_test, y\_test)

print("\nDistribution of target variable in testing set after applying SMOTE: ", y\_test\_sm.value\_counts(), sep='\n')

**1. Logistic Regression**

**Training on Balanced Training Data and Testing on Balanced Testing Data**

log\_reg = LogisticRegression()

result = log\_reg.fit(X\_train\_sm, y\_train\_sm)

y\_preds = log\_reg.predict(X\_test\_sm)

print(classification\_report(y\_test\_sm, y\_preds))

**Training on Balanced Training Data and Testing on Imbalanced Testing Data**

log\_reg = LogisticRegression()

result = log\_reg.fit(X\_train\_sm, y\_train\_sm)

y\_preds = log\_reg.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

# Doing gridsearch across 10 folds for Logistic Regression with SMOTE

log\_reg = LogisticRegression()

param\_grid = {

            'C': np.logspace(-4, 4, 50),

            'penalty': ['l2', 'none'],

            'solver': ['lbfgs', 'sag', 'saga']

            }

log\_reg\_cv = GridSearchCV(log\_reg, param\_grid, cv=10, scoring = 'recall', n\_jobs=-1, verbose=1)

log\_reg\_cv.fit(X\_train\_sm, y\_train\_sm)

print("Tuned Logistic Regression Parameters: {}".format(log\_reg\_cv.best\_params\_))

print("Best score is {}\n\n".format(log\_reg\_cv.best\_score\_))

y\_preds = log\_reg\_cv.best\_estimator\_.predict(X\_test\_sm)

print(classification\_report(y\_test\_sm, y\_preds))

Recall has increased after GridSearchCV but at the cost of accuracy and precision. The model is now able to predict more churns but at the cost of predicting more false positives. This tradeoff is acceptable as the cost of losing a potential customer is much higher than the cost of sending a promotional offer to a customer who is not likely to churn.

**2. Random Forest**

rf = RandomForestClassifier()

result = rf.fit(X\_train\_sm, y\_train\_sm)

y\_preds = rf.predict(X\_test\_sm)

print(classification\_report(y\_test\_sm, y\_preds))

# Tuning Hyperparameters of RF

rf = RandomForestClassifier()

param\_grid = {

            'n\_estimators': [100, 200, 300, 400, 500],

            'max\_depth': [3, 4, 5, 6, 7, 8, 9, 10],

            'criterion' :['gini', 'entropy']

            }

rf\_cv = GridSearchCV(rf, param\_grid, cv=10, scoring = 'recall', n\_jobs=-1, verbose=1)

rf\_cv.fit(X\_train\_sm, y\_train\_sm)

print("Tuned Random Forest Parameters: {}".format(rf\_cv.best\_params\_))

print("Best score is {}\n\n".format(rf\_cv.best\_score\_))

y\_preds = rf\_cv.best\_estimator\_.predict(X\_test\_sm)

print(classification\_report(y\_test\_sm, y\_preds))

**3. XGBoost**

# Tuning Hyperparameters of XGB with SMOTE

xgb = XGBClassifier()

param\_grid = {

            'n\_estimators': [100, 200, 300, 400, 500],

            'max\_depth': [3, 4, 5, 6, 7, 8, 9, 10],

            'learning\_rate': [0.01, 0.05, 0.1, 0.15, 0.2]

            }

xgb\_cv = GridSearchCV(xgb, param\_grid, cv=10, scoring = 'recall', n\_jobs=-1, verbose=1)

xgb\_cv.fit(X\_train\_sm, y\_train\_sm)

print("Tuned XGB Parameters: {}".format(xgb\_cv.best\_params\_))

print("Best score is {}\n\n".format(xgb\_cv.best\_score\_))

y\_preds = xgb\_cv.best\_estimator\_.predict(X\_test\_sm)

print(classification\_report(y\_test\_sm, y\_preds))

XGBoost seems to be performing the best as it has the most consistent recall and accuracy scores.

Contract\_Month-to-Month has the highest feature importance. This is because the customers who have a month-to-month contract are more likely to churn as compared to customers who have a one-year or two-year contract, since it is easier to terminate a contract on a monthly basis. This is in stark contrast to the Logistic Regression model where the feature importance of Contract\_Month-to-Month was much lower and the feature importance of tenure was much higher. However, XGBoost gives more consistent performance on the test data as compared to Logistic Regression, indicating that it is a better model for this dataset.

**Refining the model**

from sklearn.model\_selection import StratifiedShuffleSplit, StratifiedKFold, GridSearchCV

from sklearn.metrics import classification\_report

from xgboost import XGBClassifier

from imblearn.over\_sampling import SMOTE

import matplotlib.pyplot as plt

# Implementing StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n\_splits=1, test\_size=0.3, random\_state=123)

for train\_idx, test\_idx in sss.split(X, y):

    X\_train, X\_test = X.iloc[train\_idx], X.iloc[test\_idx]

    y\_train, y\_test = y.iloc[train\_idx], y.iloc[test\_idx]

print("Distribution of target variable in training set: ", y\_train.value\_counts())

# Applying SMOTE only on training data

sm = SMOTE(random\_state=123)

X\_train\_sm, y\_train\_sm = sm.fit\_resample(X\_train, y\_train)

print("Distribution of target variable in training set after SMOTE: ", y\_train\_sm.value\_counts())

# Expanding the hyperparameters for XGB

xgb = XGBClassifier()

param\_grid = {

    'n\_estimators': [100, 200, 300],

    'max\_depth': [4, 5, 6],

    'learning\_rate': [0.01, 0.05, 0.1],

    'min\_child\_weight': [1, 5, 7],

    'gamma': [0, 0.1, 0.2],

    'subsample': [0.6, 0.8, 1.0],

    'colsample\_bytree': [0.6, 0.8, 1.0]

}

# Using StratifiedKFold for cross-validation

skf = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=123)

xgb\_cv = GridSearchCV(xgb, param\_grid, cv=skf, scoring='recall', n\_jobs=-1, verbose=1)

xgb\_cv.fit(X\_train\_sm, y\_train\_sm)

print("Tuned XGB Parameters: {}".format(xgb\_cv.best\_params\_))

print("Best score is {}\n".format(xgb\_cv.best\_score\_))

y\_preds = xgb\_cv.best\_estimator\_.predict(X\_test)

print(classification\_report(y\_test, y\_preds))

# Plot feature importances

plt.figure(figsize=(15,8))

feat\_importances = pd.Series(xgb\_cv.best\_estimator\_.feature\_importances\_, index=X.columns)

feat\_importances.nlargest(15).plot(kind='barh')

plt.show()

**Model Ensembling**

from sklearn.model\_selection import StratifiedShuffleSplit, StratifiedKFold, GridSearchCV

from sklearn.metrics import classification\_report, precision\_recall\_curve, auc

from sklearn.feature\_selection import RFE

from xgboost import XGBClassifier

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from imblearn.over\_sampling import SMOTE

import matplotlib.pyplot as plt

# Implementing StratifiedShuffleSplit

sss = StratifiedShuffleSplit(n\_splits=1, test\_size=0.3, random\_state=123)

for train\_idx, test\_idx in sss.split(X, y):

    X\_train, X\_test = X.iloc[train\_idx], X.iloc[test\_idx]

    y\_train, y\_test = y.iloc[train\_idx], y.iloc[test\_idx]

# Applying SMOTE only on training data

sm = SMOTE(random\_state=123)

X\_train\_sm, y\_train\_sm = sm.fit\_resample(X\_train, y\_train)

# Feature Selection using Recursive Feature Elimination

rfe = RFE(estimator=XGBClassifier(), n\_features\_to\_select=15)

X\_train\_sm\_rfe = rfe.fit\_transform(X\_train\_sm, y\_train\_sm)

X\_test\_rfe = rfe.transform(X\_test)

# Model Ensembling

xgb = XGBClassifier()

rf = RandomForestClassifier()

voting\_classifier = VotingClassifier(estimators=[

    ('xgb', xgb), ('rf', rf)

], voting='soft')

param\_grid = {

    'xgb\_\_n\_estimators': [100, 200],

    'xgb\_\_max\_depth': [4, 5],

    'xgb\_\_learning\_rate': [0.05, 0.1],

    'xgb\_\_gamma': [0, 0.1],

    'xgb\_\_subsample': [0.8, 1.0],

    'xgb\_\_colsample\_bytree': [0.8, 1.0],

    'xgb\_\_reg\_lambda': [0.5, 1],

    'rf\_\_n\_estimators': [100, 200],

    'rf\_\_max\_depth': [5, 10]

}

# Using StratifiedKFold for cross-validation

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=123)

grid\_search = GridSearchCV(voting\_classifier, param\_grid, cv=skf, scoring='recall', n\_jobs=-1, verbose=1)

grid\_search.fit(X\_train\_sm\_rfe, y\_train\_sm)

y\_probs = grid\_search.best\_estimator\_.predict\_proba(X\_test\_rfe)[:,1]

# Post-processing: Adjusting classification threshold

precision, recall, thresholds = precision\_recall\_curve(y\_test, y\_probs)

threshold = thresholds[np.argmax(2\*(precision\*recall)/(precision+recall))]  # F1 optimized threshold

y\_preds = [1 if prob > threshold else 0 for prob in y\_probs]

print(classification\_report(y\_test, y\_preds))

# AUC of the precision-recall curve

auc\_score = auc(recall, precision)

print(f"AUC of Precision-Recall Curve: {auc\_score:.2f}")

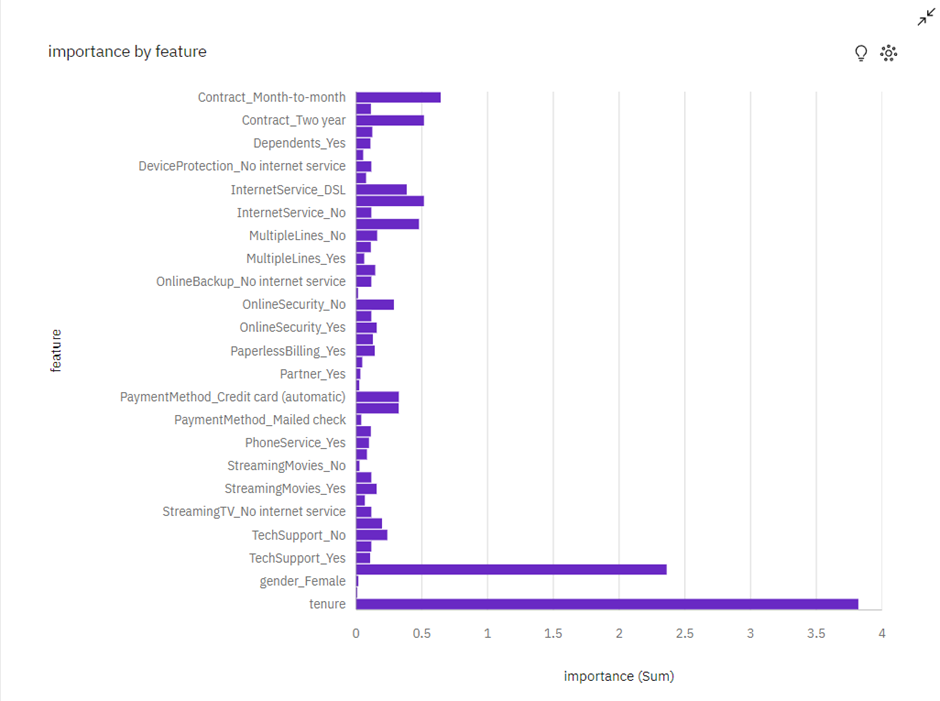
**Overall Model Performance:**

* The ensemble model exhibits good performance and has an accuracy of 77%. This means it correctly classifies approximately 77% of the instances in the dataset.
* For the churn prediction, the model demonstrates a reasonable recall of 0.70. This suggests that it is effective at identifying a good proportion of actual churn cases.
* The model performs better at identifying non-churn cases as indicated by a high precision of 0.88 and a recall of 0.80. This means it effectively distinguishes customers who are not likely to churn, with a low rate of false positives.

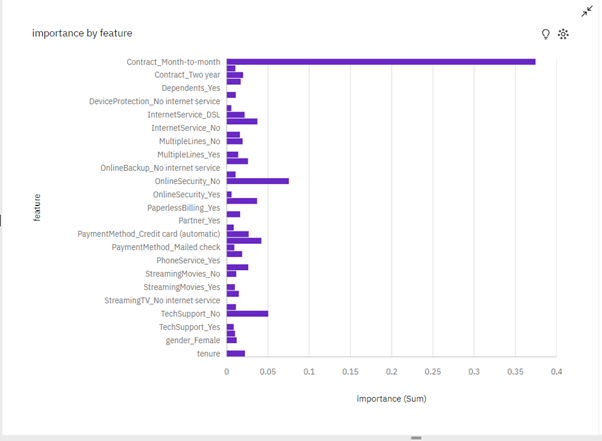
**Cognos Analytics:**

The important features influencing churn prediction were found using various machine learning algorithms and is visualised using **IBM Cognos**.

Logistic regression feature importance



Random forest & XGBoost feature importance



XGB and RF show almost identical feature importance

**Customer churn prediction** can indirectly help website owners improve user experience, especially if the website is the primary interface for a subscription-based or service-oriented business. Here's how the insights can contribute to an enhanced user experience:

**Customer Segmentation**:

Churn prediction often identify different user segments based on their likelihood of churning. Website owners can leverage these segments to customize the user experience for each group. Tailoring content, promotions, and recommendations to the specific needs and preferences of each segment can improve user satisfaction.

**Personalized Recommendations**:

Churn prediction can reveal patterns in user behavior that precede churn. By analysing these patterns, website owners can offer personalized recommendations or incentives to retain users. For example, if the model shows that users who engage with a specific type of content are less likely to churn, the website can prioritize showing that content to at-risk users.

**Content Optimization**:

Understanding the types of content or features that lead to churn can guide website owners in content optimization efforts. They can focus on improving or expanding high-churn-risk areas, making the website more engaging and valuable to users.

**Performance Optimization**:

High page load times, errors, or website performance issues can frustrate users and lead to churn. Insights from the model can highlight such performance-related problems, prompting website owners to optimize the website for faster load times and reliability.

**Conclusion**:

Thus we have built a predictive model in python using various machine learning algorithms which predicts customer churn based on the important features influencing the churn and different visualisations were carried out in IBM Cognos to analyse the import features.