Repository for data and code

Code & data repository

Problem Statement

In the telecom industry, customers can choose from multiple service providers and actively switch from one operator to another. In this highly competitive market, the telecommunications industry experiences an average of 15-25% annual churn rate. Given the fact that it costs 5-10 times more to acquire a new customer than to retain an existing one, customer retention has now become even more important than customer acquisition. For many incumbent operators, retaining highly profitable customers is the number one business goal.
To reduce customer churn, telecom companies need to **predict which customers are at high risk of churn.**

In this project, customer-level data of a leading telecom firm is analyzed, built predictive models to identify customers at high risk of churn, and identify the main indicators of churn. The recommendations
are given, and different models compared using confusion matrix, and the best model is selected. The feature engineering is also done using different mathematical procedures.

Understanding and defining churn

Usage-based churn: Customers who have not done any usage, either incoming or outgoing - in terms of calls, internet etc. over a period of time.

The project is about predicting the potential churn of customers and identifying the high risk customers who has higher probability of leaving the telecom service provider.

Modelling and the procedure to select the best model

The predictive model serves the following two purposes -

It will be used to **predict whether a high-value customer will churn**, in near future (i.e. churn phase). By knowing this, the company can act steps such as providing special plans, discounts on recharge etc.

It will be used to identify important variables that are strong predictors of churn. These variables may also indicate why customers choose to switch to other networks. Here there are large numbers of attributes, and thus **dimensionality reduction technique such as PCA** is used and then the classification model is built. Different models are trained and are tuned using **hyperparameters**.

And then the models are evaluated using appropriate **evaluation metrics**. Note that it is more important to identify churners than the non-churners accurately - choose an appropriate evaluation metric that reflects this business goal.

Finally, **choose a model based on some evaluation metric**.

The above model will only be able to achieve one of the two goals - to predict customers who will churn. We can't use the above model to identify the important features for churn. That's because PCA usually creates components that are not easy to interpret.

Therefore, another model is built with the main objective of identifying important predictor attributes that help the business understand indicators of churn. A good choice to identify important variables is a logistic regression model or a model from the tree family. In the case of logistic regression, make sure to handle multi-collinearity.

After identifying important predictors, they are displayed visually using plots, summary tables.

Finally, strategies are recommended to manage customer churn based on the observations.

Test and Train split

Test size is taken as 25% of the total data set. 4 random states are used.

Train Test split

```
# divide data into train and test
X = churn_filtered.drop("churn", axis = 1)
y = churn_filtered.churn
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 4, stratify = y)
```

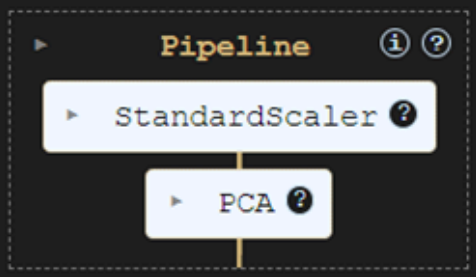
Aggregating the categorical features.

The categorical columns are aggregated to reduce the features.

PCA on the remaining features.

Principle Component Aanalysis is applied on the remaining features, to reduce the number of features further, retaining as much information as possible in the data. This helps in handling the curse of dimensionality. The correlated features are reduced and only the independent ones are kept.

```
pca.fit(X_train)
churn_pca = pca.fit_transform(X_train)
```




```
# extract pca model from pipeline
pca = pca.named_steps['pca']

# look at explainded variance of PCA components
print(pd.Series(np.round(pca.explained_variance_ratio_.cumsum(), 4)*100))
```

Outputs are collapsed ···

~ 60 components explain 90% variance

~ 80 components explain 95% variance

Logistic regression

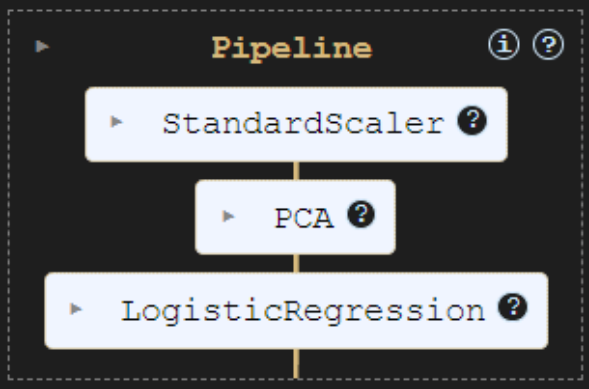
Logistic regression is applied to classify the data on churning and non churning customers. The score of the model is computed as 80%.

```
# create pipeline
PCA_VARS = 60
steps = [('scaler', StandardScaler()),
         ("pca", PCA(n_components=PCA_VARS)),
         ("logistic", LogisticRegression(class_weight='balanced'))
        ]
pipeline = Pipeline(steps)
```

[61]

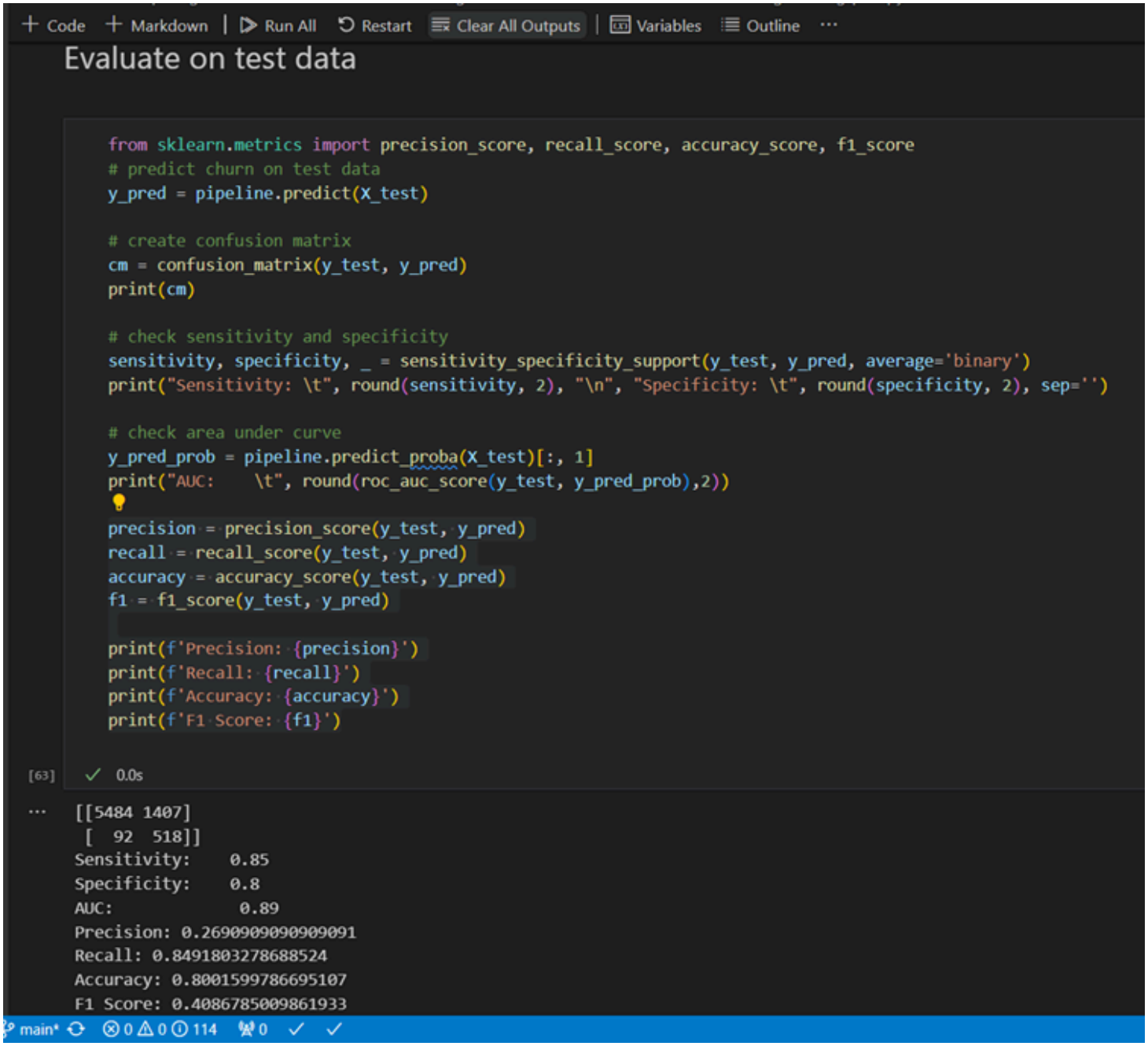
```
# fit model
pipeline.fit(X_train, y_train)

# check score on train data
pipeline.score(X_train, y_train)
```

[62]

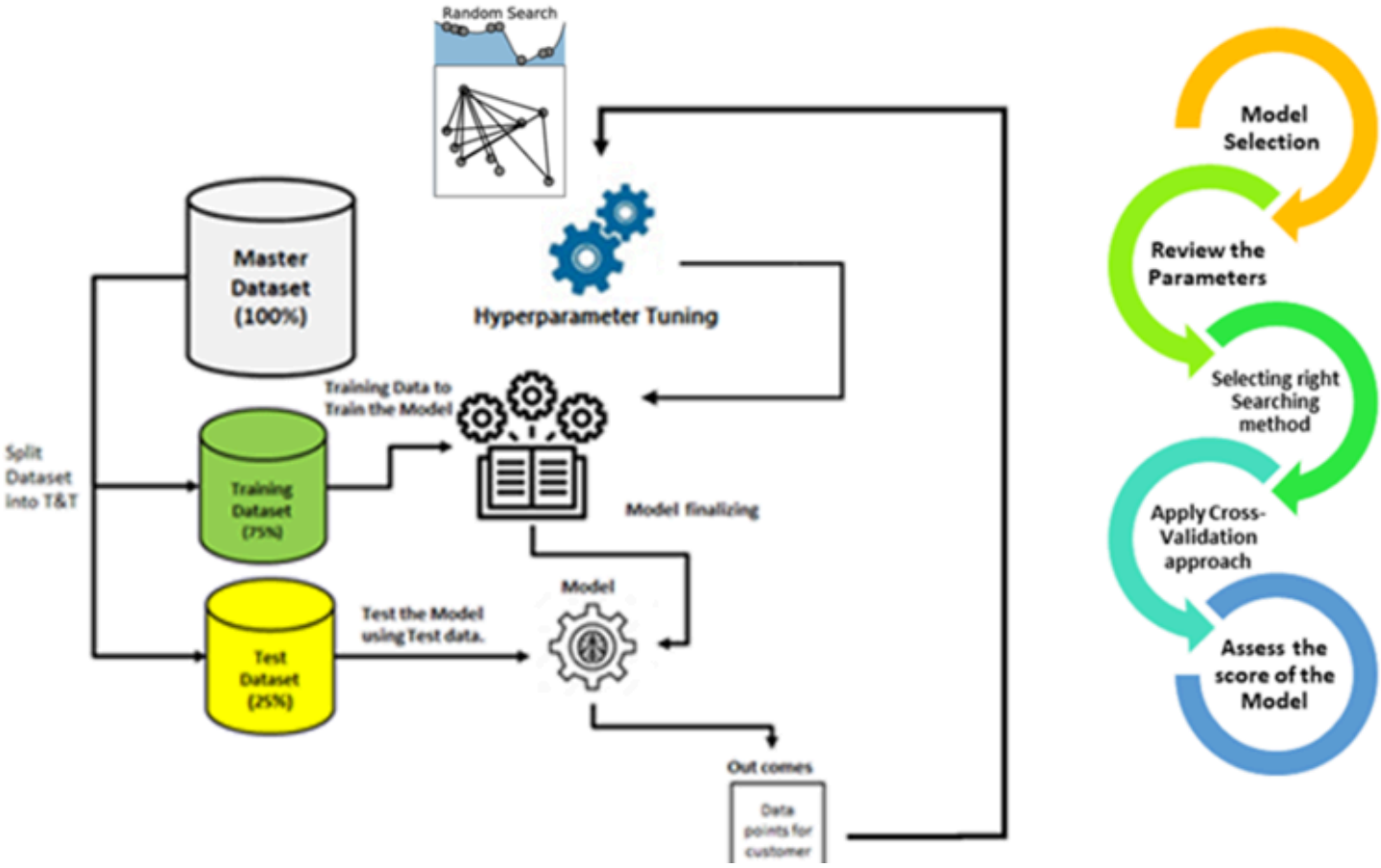



```
0.8065777777777777
```

Confusion matrix for the logistic regression on test data



Hyperparameter tuning and logistic regression



Grid search cross validation method is used to do the hyperparameter tuning. A 5 fold search is done to do the hyperparameter tuning. Following are the steps to hyper parameter tuning as depicted in the figure above --

- Select the right type of model.
- Review the list of parameters of the model and build the HP space
- Finding the methods for searching the hyperparameter space
- Applying the cross-validation scheme approach
- Assess the model score to evaluate the model

```python
# PCA
pca = PCA()

# logistic regression - the class weight is used to handle class imbalance - it adjusts the cost function
logistic = LogisticRegression(class_weight={0:0.1, 1: 0.9})

# create pipeline
steps = [("scaler", StandardScaler()),
         ("pca", pca),
         ("logistic", logistic)
        ]

# compile pipeline
pca_logistic = Pipeline(steps)

# hyperparameter space
params = {'pca__n_components': [60, 80], 'logistic__C': [0.1, 0.5, 1, 2, 3, 4, 5, 10], 'logistic__penalty': ['l1', 'l2']}

# create 5 folds
folds = StratifiedKFold(n_splits = 5, shuffle = True, random_state = 4)

# create gridsearch object
model = GridSearchCV(estimator=pca_logistic, cv=folds, param_grid=params, scoring='roc_auc', n_jobs=-1, verbose=1)
```

[65] Python

```python
# fit model
model.fit(X_train, y_train)
```

[66] Python

```
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

Model performance post hyperparameter tuning for logistic regression.

```
# predict churn on test data
y_pred = model.predict(X_test)

# create confusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# check sensitivity and specificity
sensitivity, specificity, _ = sensitivity_specificity_support(y_test, y_pred, average='binary')
print("Sensitivity: \t", round(sensitivity, 2), "\n", "Specificity: \t", round(specificity, 2), sep='')

# check area under curve
y_pred_prob = model.predict_proba(X_test)[:, 1]
print("AUC:    \t", round(roc_auc_score(y_test, y_pred_prob),2))

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Accuracy: {accuracy}')
print(f'F1 Score: {f1}')
```

[69] ✓ 0.0s

```
[[5770 1121]
 [ 104  506]]
Sensitivity:    0.83
Specificity:    0.84
AUC:             0.9
Precision: 0.3110018438844499
Recall: 0.8295081967213115
Accuracy: 0.8366884415411279
F1 Score: 0.4523915958873491
```

As shown above the performance of the model has improved, and the AUC is 90%.

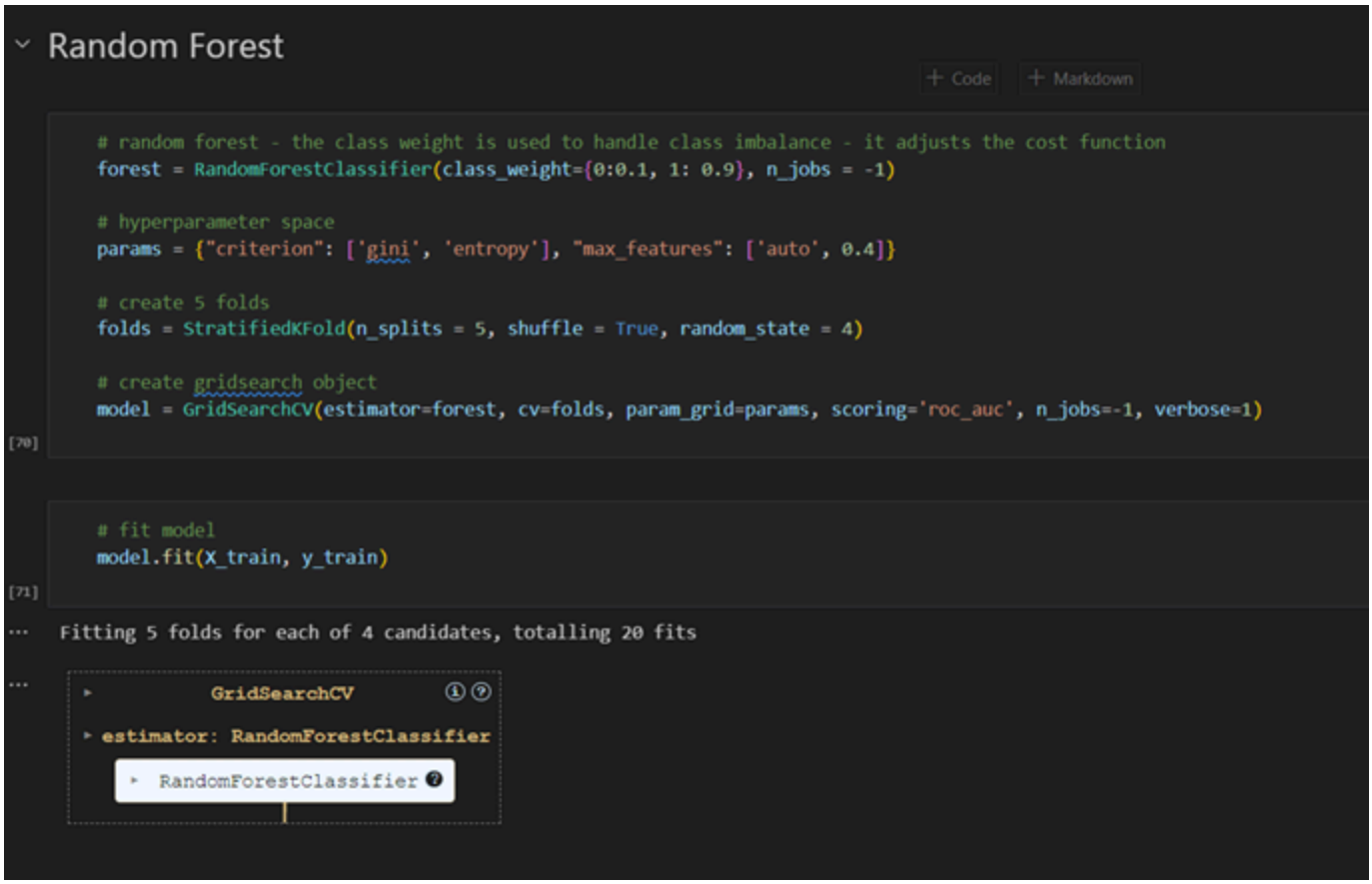
Scorecard for Logistic regression improvements

Logistic regression	Before hyperparameter	Post-hyperparameter
Sensitivity	85%	**83%**
Specificity	80%	**84%**
AUC	89%	**90%**

Logistic regression	Before hyperparameter	Post-hyperparameter
Precision	27%	**31%**
Recall	85%	**83%**
Accuracy	80%	**84%**
F1 score	41%	**45%**
AUC	89%	**90%**

The scorecard of the model performance of the logistic regression in the above table shows that the performance of the model has drastically improved after hyper parameter tuning.

Decision tree -- random forest after hyper parameter tuning



Model performance -- random forest

```
# predict churn on test data
y_pred = model.predict(X_test)

# create onfusion matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)

# check sensitivity and specificity
sensitivity, specificity, _ = sensitivity_specificity_support(y_test, y_pred, average='binary')
print("Sensitivity: \t", round(sensitivity, 2), "\n", "Specificity: \t", round(specificity, 2), sep='')

# check area under curve
y_pred_prob = model.predict_proba(X_test)[:, 1]
print("AUC:    \t", round(roc_auc_score(y_test, y_pred_prob),2))

precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print(f'Precision: {precision}')
print(f'Recall: {recall}')
print(f'Accuracy: {accuracy}')
print(f'F1 Score: {f1}')
```

[73] ✓ 0.1s

```
[[6778  113]
 [ 303  307]]
Sensitivity:    0.5
Specificity:    0.98
AUC:            0.93
Precision: 0.7309523809523809
Recall: 0.5032786885245901
Accuracy: 0.9445407279029463
F1 Score: 0.596116504854369
```

Comparison of Classification Model Recommendation

Hence the best model is PCA along with logistic regression, as following are the model performance metrics after doing hyperparameter tuning -

Metrics after hyper param	Logistic Regression	Radom forest decision
Sensitivity	83%	**50%**
Specificity	84%	**98%**
AUC	90%	**93%**
Precision	31%	**73%**

Metrics after hyper param	Logistic Regression	Radom forest decision
Recall	83%	**50%**
Accuracy	84%	**94%**
F1 score	45%	**60%**
AUC score	90%	**93%**

Model recommendation

The Random Forest fares better than Logistic regression on all fronts except Recall and Sensitivity. Based on the scenarios at hand, the model can be chosen. In all cases, where the sensitivity of the data is not crucial, Random forest appears good. But the recall of the Random forest is way lesser than that of the Logistic regression. Recall is the ability to predict correctly when an event is classified as positive.
This scenario is important in this business context of predicting the churn of customers when they will churn out. So, for this business scenario**[, the recommended model needs to be the Logistic regression
model after hyperparameter tuning.]{.underline}**