To what extent does hyperparameter tuning affect the cross-validation score of the random forest algorithm in predicting customer churn in a subscription-based telecom service?

Vivan Desai

I Introduction

Over the past few years, the telecom industry consisting of internet service providers, mobile network operators have seen major growth by increasing the quality of their subscription based products. A crucial challenge that the industry faces is customer churn, which is when customers discontinue their subscription. An example is a customer discontinuing their mobile phone service subscription. This problem is of great significance because it has an impact on the companies profit trajection and to develop improved business strategies (paddle). The phenomenon of customer churn is accounted for within businesses however, there is a gap in predicting whether or not a customer will discontinue their subscription. Even with the use of various machine learning algorithms to predict customer churn based on past data from telecom service providers, there is no established conclusion regarding which algorithm is best performing a customer churn prediction. Machine learning is a branch of computer science which allows computers to make predictions and decisions without being programmed. The two ways by which machine learning models can be developed is unsupervised, where the model by itself determines a pattern. Rather methods like supervised learning use already labeled data (it gives a pattern to work off on) to train machine learning models to predict or classify data (IBM). The objective of this investigation is to compare how hyperparameter tuning a random forest algorithm affects the performance metrics (accuracy, precision, recall, and F1-score) compared to a plain (unmodified) random forest model.

II Literature Review

Through this review insight can be gained about problems that could appear during the investigation, the process to acquire clean data that is not noisy. Both noisy and clean refer to the levels of cleanliness which can impact the performance of machine learning algorithms in making accurate predictions. Noisy data contains characteristics such as outliers, inconsistencies, and mislabeling. On the other hand, clean data is the opposite and as a result is always better when developing machine learning models.

From the study done by Kavitha, S. V Mohan Kumar, G. Hemanth Kumar and M. Harish, titled "Churn Prediction of Customers in Telecom Industry using Machine Learning Algorithms", I will be extracting the key elements in their methodology. These key elements include their data preprocessing techniques which is directly linked to noise removal and making the data viable for testing on the random forest model. The useful part from this research paper is part B in the methodology section where it discusses the reasons for manipulating the data into binary (0s and 1s), as a result it is easier for the computer to understand. Furthermore, the paper explores feature selection and only chooses features that have a significant impact/correlation with the target variable which is churn (whether the customer has churned or not). However, this study does not explain tools used in order to validate the reason behind their feature selection, therefore I will improve this process of feature selection using resources such as matplotlib. Through this I will be able to quantify the impact of features on the target variable allowing a more valid approach to feature selection. Furthermore, the study conducted does use the random forest model in order to predict customer churn, however I will improve on this by doing both a simple random forest model as well as a hyperparameter tuned random forest model. The ultimate goal is to improve the accuracy of 0.80 which was achieved in their study, this means 80% of the data set that was used was correct predictions (ratio of correct predictions to total instances in the data set).

III Background Information

A. Machine Learning

Mentioned previously, there are two primary methods of learning, one being supervised learning and the other is unsupervised learning. Supervised learning is when the chosen algorithm is learning from labeled data to accurately predict the corresponding target and usually is used for classification problems (International Business Machines Corporation). An example of a classification problem is spam detection and to classify whether or not the email should be placed in the spam folder or not. On the other hand, unsupervised learning is trained on unlabeled data implying there is no target variable nor class labels. The main goal from unsupervised learning is to identify relationships and patterns from the data which is unlabeled. An application of this in real life is object detection in self-driving vehicles. For the purpose of this research paper it is a complete classification problem, as the data used is already labeled and the aim is to classify whether or not a customer will churn using supervised learning. Within supervised learning, numerous algorithms exist, including neural networks, support vector machines, and linear regression. However, this investigation will focus primarily on the random forest algorithm.

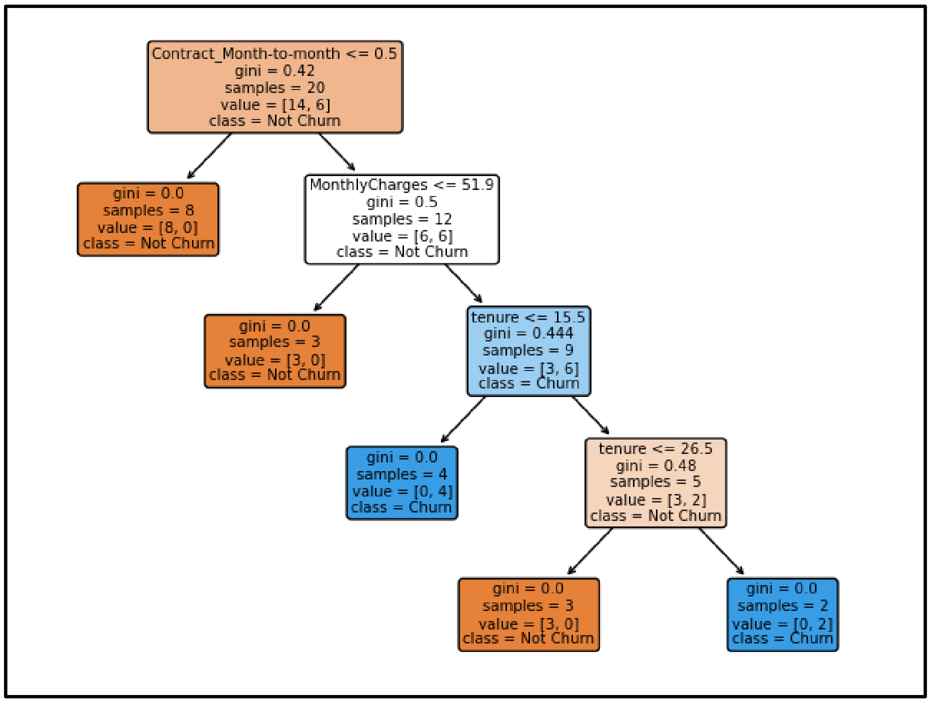
B. Decision Trees

Decision trees are an important component within the random forest algorithm, therefore playing a key role in the objective of this investigation which primarily deals with the random forest model. Decision trees are used for both classification and regression tasks, for the purpose of this investigation it relates to the classification of churn and not churn. It is visually represented using a hierarchical tree structure, beginning from the root node which only has outgoing branches to internal nodes also known as decision points which are points based on the independent variables in a data set, and lastly there are leaf/terminal nodes which represent the final outcome (Song and Lu 1). In the case of this investigation the final outcome is whether the customer has churned or not. The goal of the decision tree algorithm is to retrieve pure terminal nodes (homogeneous outcomes belonging to the same class), but as we increase the number of independent variables it is difficult to obtain pure terminal nodes as the tree becomes more complex creating more splits and therefore smaller groups of data in each node (International Business Machines Corporation). Additionally, a commonly used method in decision trees is gini impurity. Gini impurity is the probability of misclassifying the outcome of the target variable, true or false (0 and 1). It considers each feature (independent variables) and evaluates potential splits based on the values in that feature by calculating the gini impurity by using the following formula.

$$\text{Gini} = 1 - \text{i} = 1\text{C}(\text{Pi})2$$

Here 'C' represents total classes/features, 'Pi' is the probability of picking the data point with the class 'i' (Jairi). Finally, when calculating the gini impurity, it only considers which split retained the lowest value as it signifies the least impurity. This links to the research question because it is one of the parameters which will be tuned when testing the normal unmodified random forest model to the hyper parameter tuned model.

Figure 1. Decision Tree Visualized From Dataset (Created by Candidate)



In figure 1, the larger nodes in the decision tree represent the internal nodes or decision points which use the features, here we see 'MonthlyCharges' and 'TotalCharges' used. Furthermore the gini value at the decision points is greater than zero indicating that the data has to be split further, the remnant nodes indicate a gini value of 0 meaning there is no impurity.

C. Random Forest

The random forest algorithm is an ensemble learning method, involving bootstrapping and aggregation which is bagging (Bertsimas) . Random forest is made of multiple decision trees, where each tree uses a random subset of features from the data set which is called bootstrapping (Scornet 2). The final prediction is retrieved by aggregating all the outcomes from the individual decision trees, and is done by majority voting or averages depending whether it is a classification or regression task. Furthermore, the random forest algorithm has a range of hyperparameters, but for this investigation we will be considering 5 hyperparameters which are the following 'n_estimators', 'criterion', 'max_depth', 'max_features' and 'class_weight'. Each of these parameters have their own function that tunes the algorithm.

- 'n_estimators' parameter decides the number of decision trees within a random forest, as a result more trees can capture a range of patterns, minimizing the problem of overfitting.
- 'criterion' parameter is a function to measure the quality of the split, as mentioned prior in this investigation the gini impurity is one method to calculate the pureness of splits.
- 'max_depth' parameter puts a limit on how deep a decision tree can grow. The value for this parameter is extremely influential as a large number will result in being too specific in the training data and will not be able to generalize patterns in the test data.
- 'Max-features', considers the number of features which are randomly selected for each split, it is usually set to 'sqrt' or 'log2'.

- ‘class_weight’ is used to address the imbalance in the target variable values, as a result it will assign a different weight for the classes to make it balanced. (Pedregosa and Varoquaux).

When finding appropriate values for hyperparameter tuning it can significantly enhance the models performance, although optimal values for these parameters can cause problems such as overfitting (International Business Machines Corporation).

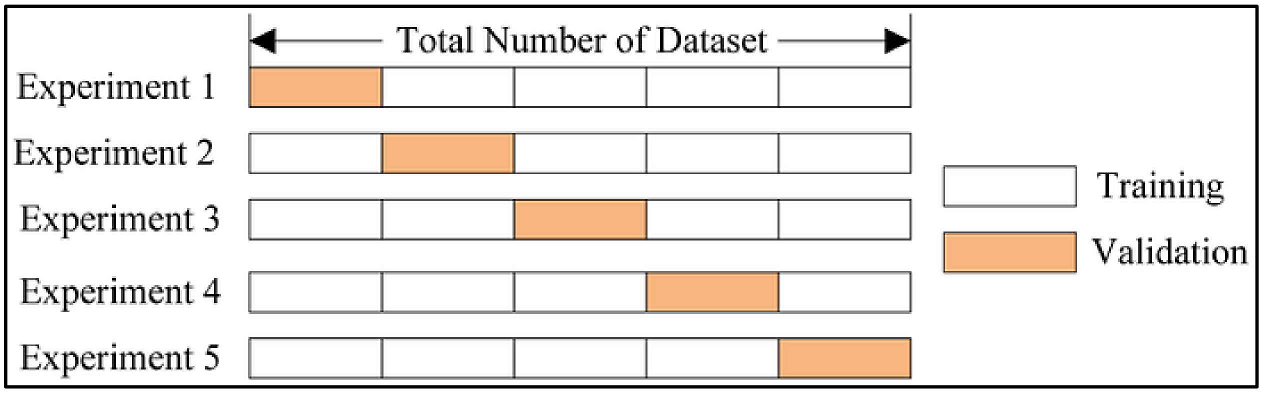
D. Cross Validation

Cross-validation is a technique used in machine learning to evaluate a model's performance and generalizability on unseen data. This investigation will use k-fold cross-validation which then can be used to retrieve a cross validation score which is the primary metric used to compare the tuned and untuned random forest model. The process is the following:

> First, the available data is split into a training set and a separate test set using a train-test split. This ensures that the final evaluation is done on unseen data. Next, the training set is further divided into k equal-sized groups/folds (Wong 2). The number of folds, k, can be adjusted within the parameters. For example, we perform a 5-fold cross-validation. The training set is split into 5 equal groups, and the process will run a total of 5 iterations (Koehrsen).
>
> During each iteration, one fold is used as the validation or test set, and the remaining k-1 folds are used as the training set. This allows the model to be

trained on different subsets of the training data in each iteration. For instance, in the first iteration, the first fold is used for validation and the remaining 4 folds are used for training. In the second iteration, the second fold becomes the validation set, and the rest are used for training, and the process carries on. This can be visualized in figure 3.

Figure 3. 5-Folds cross validation visualization (Koehrsen)



Each iteration is evaluated using accuracy which is calculated by # of Correct PredictionsTotal # of Predictions, and then the mean of each iteration is returned as the final cross validation score.

IV Methodology

A. Data Set

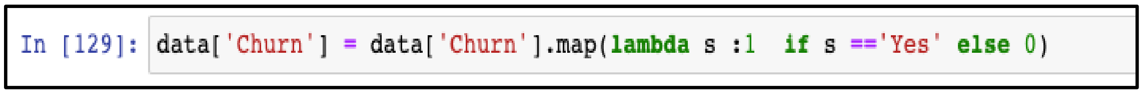
This research paper uses a dataset from Kaggle, a well-known platform that fosters data-driven research among data scientists (Wikipedia). The dataset contains information about customer demographics, behavior, and account details from a telecom service provider. It consists of 7,043 rows representing individual customers and 21 columns (features). The target column, 'churn,' indicates whether a customer has churned, with values 'yes' or 'no.' The dataset includes demographic information such as age, gender, and partnership status. It also incorporates customer account information, including charges, tenure, and payment methods. Additionally, the dataset captures if the customer has signed up for additional services like streaming, internet, tech support, online security, device protection, and multiple lines.

B. Data Preprocessing and Exploration

Data preprocessing and exploration are required steps before initializing a train test split. Preprocessing deals with cleaning the data which includes, removing any missing values (NaN), ensuring data consistency, encoding categorical values, and transforming data types if necessary. These steps as a result, helps and clean the data making it more suitable and easy to understand for the machine learning algorithms that will be used to test the data set. Similarly exploration is crucial as it allows us to gain understanding about the data set and uncover relationships between features. Through the use of visualization libraries such as Matplotlib, features and their relationships can be represented via numerous graphs, charts and most importantly heatmaps. By exploring the dataset visually, we can make informed decisions about feature selection and identify which features have the most significance on predicting the target variable. Heatmaps, in particular, can reveal the strength of relationships between features, aiding in the identification of dependencies within the features of the dataset.

1. Firstly I checked for all the unique values within the dataset using pandas nunique() function. Surprisingly, the 'tenure' column only had 73 unique values out of the 7043 possible values. Therefore I replaced all the tenure values which had 0 with the mean tenure value, to ensure that there are no extreme outliers and there is data consistency.
2. Secondly, I had to deal with data transformation particularly with the columns which were categorical. Out of the 21 features there were only 3 numerical columns being 'SeniorCitizen', 'tenure' and 'MonthyCharge'. The rest were all categorical. As a result I had to undergo one hot encoding, which changes the feature values to binary 0s and 1s. The following code is an example of how I repeated the process for each column:
3.

Figure 4. Code of One Hot Encoding Categorical Features(Created by Candidate)



```
In [129]: data['Churn'] = data['Churn'].map(lambda s :1  if s =='Yes' else 0)
```

As a result from the code snippet in figure 4, if the value form the 'Churn' column is 'yes' it would be replaced with '1' else it would be '0'. I did not use the pandas get_dummies() function as some features had 2 repetitive unique values such as 'yes', 'no' and 'no internet'.

4. Finally once all the features were dealt with, I now checked for missing values within the data set. This can be done by 'data.isnull().sum()', which retrieves how many null values are present in each feature. The 'TotalCharges' feature had 11 missing values, as a result I dropped all the rows with the missing values. Now the data set is ready to

be plotted onto a heat map to see the direction and strengths of the features and their relationships to each other.

Figure 5. Heatmap of Cleaned Data Set (Created by Candidate)

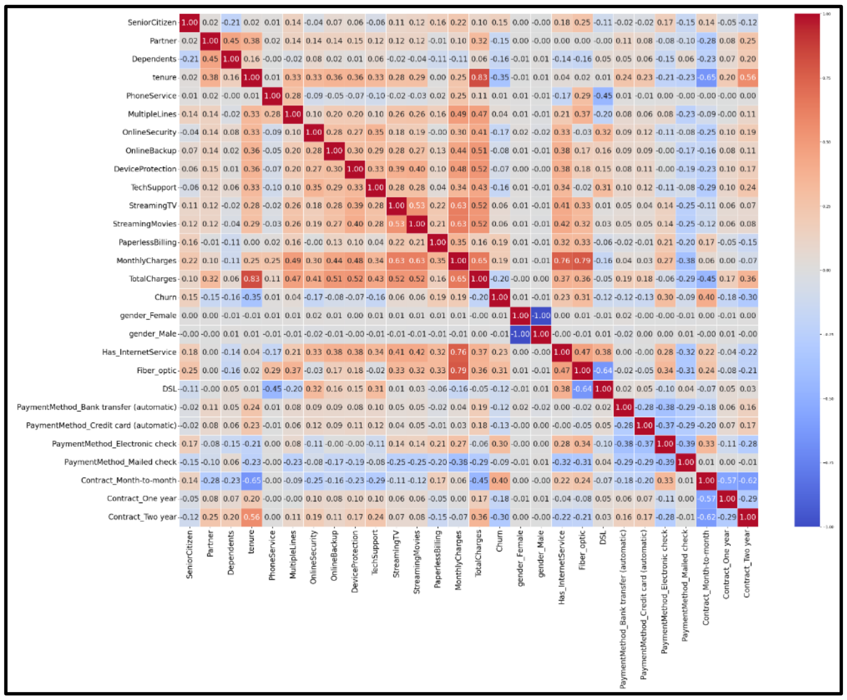


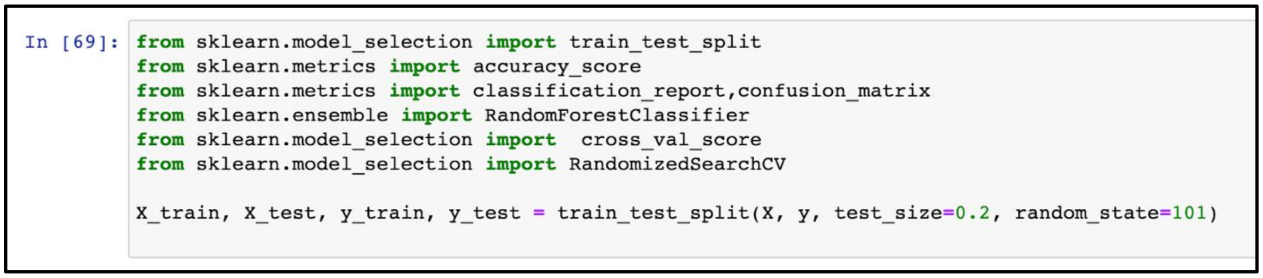
Figure 5 illustrates the correlation between the features in the dataset, where the main feature we should be considering is ‘Churn’. The values are the Pearson correlation coefficient, where we see that a contract that is month to month displays a strong correlation to churn (0.40), relative to the other features. Similarly customers who use fiber optic internet connection show a positive correlation to churn (0.33) as compared to those who use a DSL connection. Through this I got a general idea of the relationships between the features, particularly the target variable.

C. Model Building and Hyperparameter Tuning

The data has been cleaned, preprocessed, explored and now has to be trained and tested on the random forest model. A baseline random forest model will be created without any hyperparameter tuning, as it will be used as a reference point when evaluating the second model which will be tuned on 5 hyperparameters.

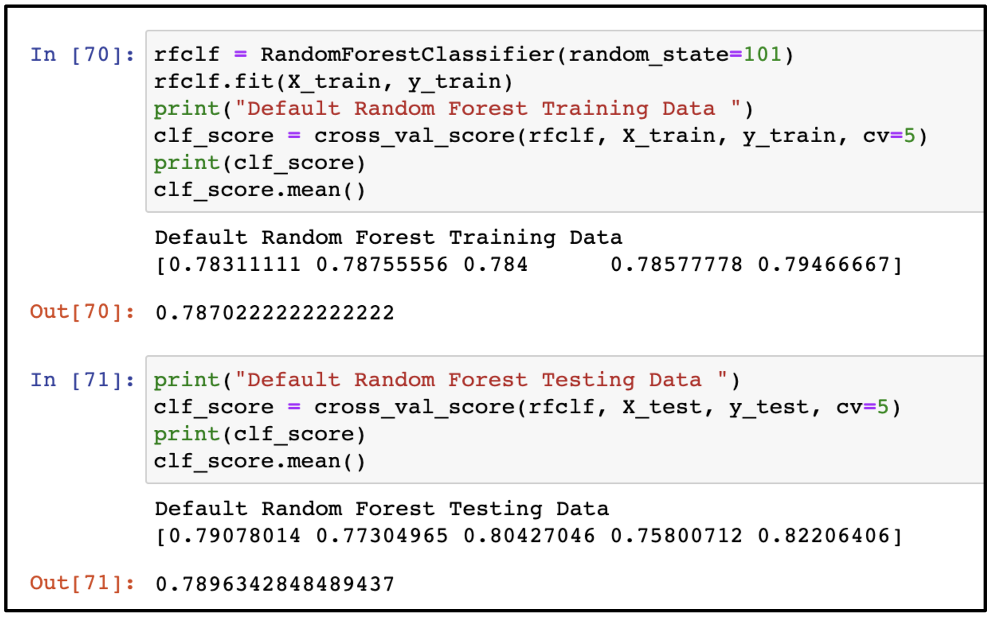
Firstly, the dataset is divided into two separate variables: 'x' and 'y'. 'y' represents the target column, while 'x' contains the remaining features. This step is crucial as it separates the input features from the target variable, enabling us to build a predictive model. To assess the model's performance accurately, we employ a train-test split using Scikit-learn's train_test_split() function. In figure 6, by specifying a test size of 0.2, we allocate 20% of the data for testing, while the remaining 80% is used for training. This division allows us to evaluate the model's ability to generalize and gauge patterns on unseen data. To ensure consistency and fair comparisons, a fixed random state value of 101 is utilized throughout the model building process.

Figure 6. Implementation of Train Test Split (Created by Candidate)


```
In [69]: from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report,confusion_matrix
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import  cross_val_score
         from sklearn.model_selection import RandomizedSearchCV

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

After the separation of 'x' and 'y' variables into their training and testing datasets, it is time to fit the baseline model, without hyperparameter tuning. In the presented implementation (Figure 7), an instance of the Random Forest Classifier class is created and named 'rfclf', with a random state value of 101 assigned to ensure consistency. The Random Forest Classifier is then fitted on the training set, which is specified as a parameter. Furthermore, to evaluate the model's performance we employ a cross validation fold to the training set dividing it into 5 equal folds, allowing for an assessment of the model's accuracy. In figure 7, we see that the baseline model achieves an accuracy score of 78.7% (0.787) on the training set. Now we proceed to test its performance on unseen data/the testing set. By using the mean of the cross-validation scores obtained from the testing set, the baseline model achieves an accuracy score of 79.0% (0.789) on the testing set. Finally we have a reference point of a baseline random forest model which can be used to compare against the hyper tuned model.

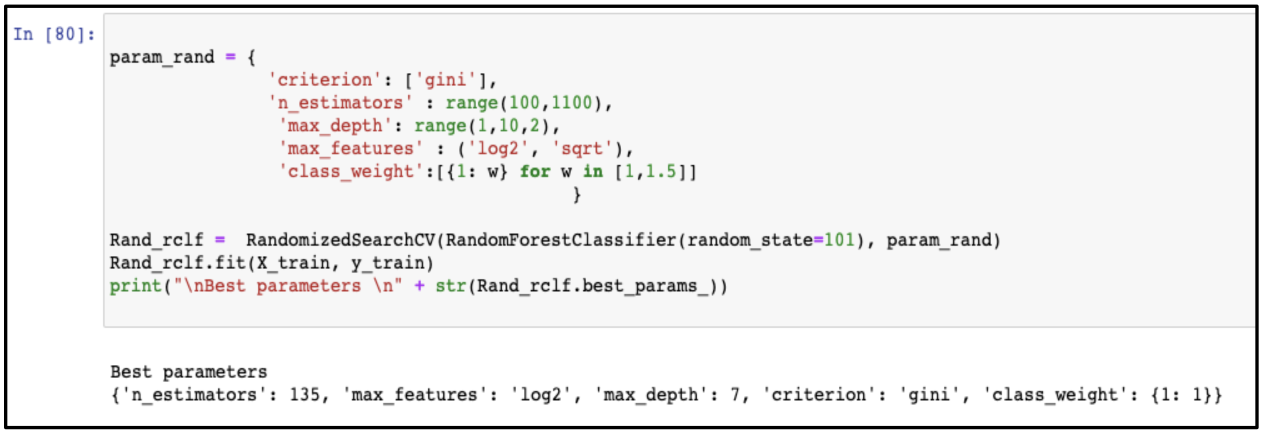
Figure 7. Baseline Model Building (Created by Candidate)



```
In [70]: rfclf = RandomForestClassifier(random_state=101)
         rfclf.fit(X_train, y_train)
         print("Default Random Forest Training Data ")
         clf_score = cross_val_score(rfclf, X_train, y_train, cv=5)
         print(clf_score)
         clf_score.mean()

         Default Random Forest Training Data
         [0.78311111 0.78755556 0.784      0.78577778 0.79466667]

Out[70]: 0.7870222222222222

In [71]: print("Default Random Forest Testing Data ")
         clf_score = cross_val_score(rfclf, X_test, y_test, cv=5)
         print(clf_score)
         clf_score.mean()

         Default Random Forest Testing Data
         [0.79078014 0.77304965 0.80427046 0.75800712 0.82206406]

Out[71]: 0.7896342848489437
```

Moving on to creating the hyper tuned model, we need to establish the best parameters that are best suited for the random forest model. To begin, a dictionary named 'param_grid' is created, which contains potential values for the selected hyperparameters: 'criterion', 'n_estimators', 'max_depth', 'max_features', and 'class_weight'. Each hyperparameter is assigned a range or a list of possible values that will be explored and tested on the training set. Next, an instance of the randomized search class is created and named 'Rand_rclf'. This instance contains the Random Forest Classifier, the random state value, and the 'param_grid' dictionary as the parameters. The randomized search algorithm randomly selects combinations of hyperparameters from 'param_grid' and evaluates their performance to determine the most suitable combination. Subsequently, 'Rand_rclf' is fitted to the training data, consisting of the 'x' and 'y' variables. In figure 8, we finally see the best parameters which are '{'n_estimators': 135, 'max_features': 'log2', 'max_depth': 7, 'criterion': 'gini', 'class_weight': {1: 1}}'.

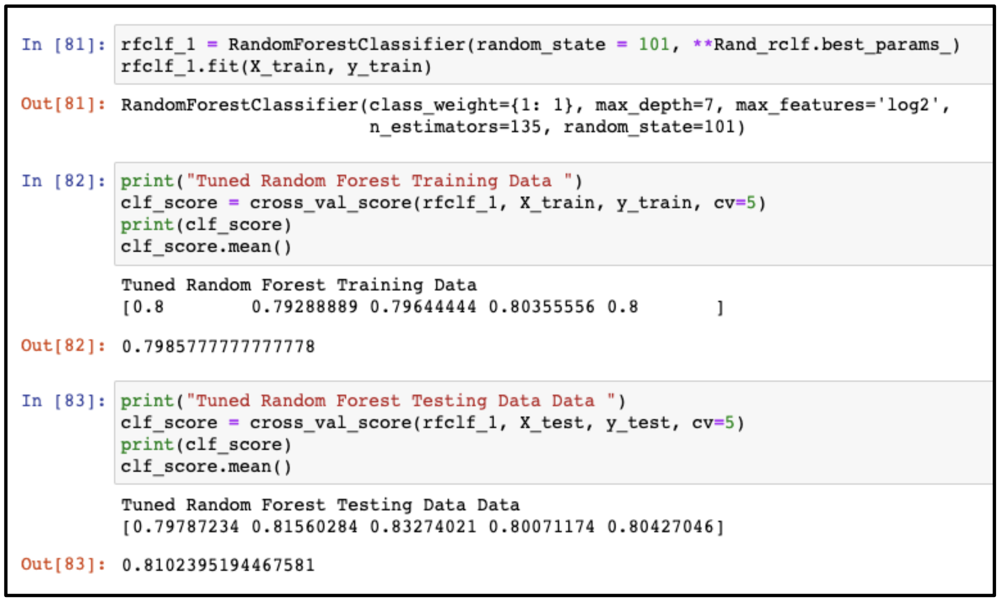
Figure 8. Implementation of Randomized Grid Search (Created by Candidate)



```
In [80]:
param_rand = {
                'criterion': ['gini'],
                'n_estimators' : range(100,1100),
                 'max_depth': range(1,10,2),
                 'max_features' : ('log2', 'sqrt'),
                 'class_weight':[{1: w} for w in [1,1.5]]
                                  }

Rand_rclf =  RandomizedSearchCV(RandomForestClassifier(random_state=101), param_rand)
Rand_rclf.fit(X_train, y_train)
print("\nBest parameters \n" + str(Rand_rclf.best_params_))


Best parameters
{'n_estimators': 135, 'max_features': 'log2', 'max_depth': 7, 'criterion': 'gini', 'class_weight': {1: 1}}
```

Having determined the best combination of hyperparameters, we proceed to create the hyper-tuned model and evaluate its performance. Similar to the process of creating the baseline model, we instantiate a new instance of the Random Forest Classifier named 'rfclf_1' with a random state value of 101 and the best parameters that were identified from the randomized grid search. Subsequently, 'rfclf_1' is fitted to the training data. To ensure a fair comparison with the baseline model, a cross-validation technique is employed. The training data is divided into 5 equal folds, allowing for an evaluation of the model's accuracy on each fold. By averaging the accuracy scores across the folds, we get the mean accuracy of 79.8% (0.789) for the training data. This measure provides an indication of the model's performance on the training set. Finally, we employ the testing/unseen data to evaluate the 'rfclf_1' model. Using the same cross-validation approach, we obtain the mean accuracy score of 81.0% (0.810) across the folds which can be seen in figure 9.

Figure 9. Hyperparamter Tuned Model Implementation and Results (Created by Candidate)



```
In [81]: rfclf_1 = RandomForestClassifier(random_state = 101, **Rand_rclf.best_params_)
         rfclf_1.fit(X_train, y_train)

Out[81]: RandomForestClassifier(class_weight={1: 1}, max_depth=7, max_features='log2',
                                n_estimators=135, random_state=101)

In [82]: print("Tuned Random Forest Training Data ")
         clf_score = cross_val_score(rfclf_1, X_train, y_train, cv=5)
         print(clf_score)
         clf_score.mean()

         Tuned Random Forest Training Data
         [0.8        0.79288889 0.79644444 0.80355556 0.8       ]

Out[82]: 0.7985777777777778

In [83]: print("Tuned Random Forest Testing Data Data ")
         clf_score = cross_val_score(rfclf_1, X_test, y_test, cv=5)
         print(clf_score)
         clf_score.mean()

         Tuned Random Forest Testing Data Data
         [0.79787234 0.81560284 0.83274021 0.80071174 0.80427046]

Out[83]: 0.8102395194467581
```

V Results

The aim for this investigation is comparing two random forest classifier models, one being hyperparameter tuned and the other being a baseline/default. The models were evaluated using a cross validation score and taking the average accuracy from each fold.

Dataset	**Cross Validation Score**
Baseline Training Set	**0.7870**
Baseline Testing Set	**0.7896**
Hyperparamter Tuned Training set	**0.7985**
Hyperparamter Tuned Testing set	**0.8102**

A. Training Set Performance

On the training set, the hyperparameter tuned model achieved a slightly higher cross validation score of 0.7985 as compared to the baseline model which scored 0.7870.

Although it is only slightly higher, it indicates the hyperparameter tuned model was better at reading patterns on the training data set to a small extent, as the difference between the cross validation scores are minimal. This could be due to the data quality and quantity as more data will help the model learn better, furthermore and most importantly the optimal parameters. Hyperparameter tuning is treated with a marginal nature, this means when the parameters are too tuned it can cause small marginal gains in the cross validation score.

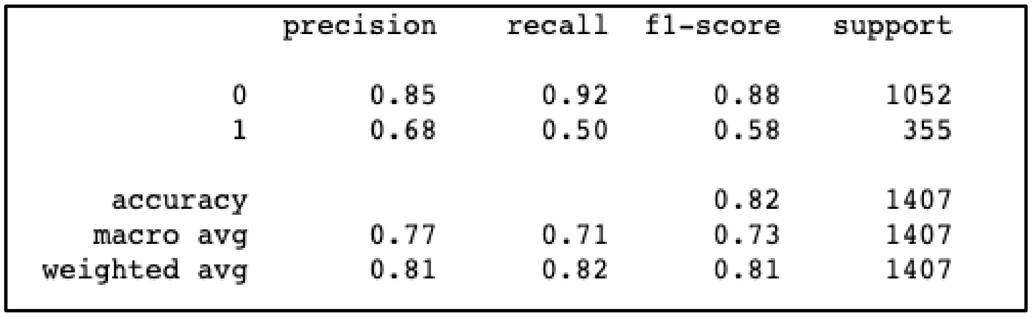
B. Testing Set Performance

When evaluating the testing set, the hyperparameter tuned model performed superiorly with an cross validation score of 0.8102 compared to the baseline model which achieved an cross validation score of 0.7896. This indicates the baseline model does have a limit to what it can achieve in terms of its cross validation score and that is due to its no tuning of parameters. However the tuned model exhibited better performance, and the difference in the cross validation scores is significant, as it illustrates the improvement by using a tuned model.

C. Literature Review Random Forest Performance

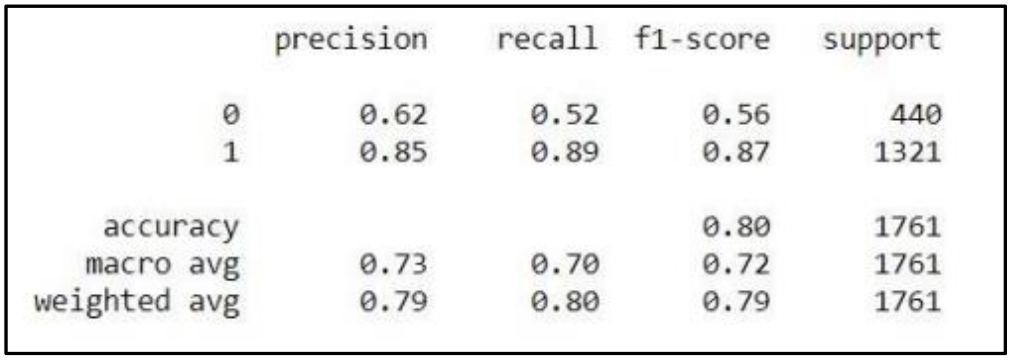
Referring back to (Kumar et. al), their study achieved an accuracy of 0.80 on the random forest model, where no hyperparameters were optimized nor did the study implement a cross validation technique. Their study used a classification report to measure the accuracy of the random forest, therefore to ensure it is a fair comparison I implemented a classification report. We see from figures 10 and 11, my hyperparameter tuning random forest model archives a higher score across all accuracy, precision, recall and f1-score.

Figure 10. Hyperparameter Tuned Classification Report on Random Forest (Created by Candidate)



```
              precision    recall  f1-score   support

           0       0.85      0.92      0.88      1052
           1       0.68      0.50      0.58       355

    accuracy                           0.82      1407
   macro avg       0.77      0.71      0.73      1407
weighted avg       0.81      0.82      0.81      1407
```

Figure 11. Literature Reviews Classification Report on Random Forest (Kumar et al. 3)



```
              precision    recall  f1-score   support

           0       0.62      0.52      0.56       440
           1       0.85      0.89      0.87      1321

    accuracy                           0.80      1761
   macro avg       0.73      0.70      0.72      1761
weighted avg       0.79      0.80      0.79      1761
```

Overall it is visible that using optimized parameters in the random forest model will result in enhanced performance across all metrics. Specifically, the accuracy has increased by 2.5% when hyperparameter tuning the model. This 2.5% increase is substantial however, when factoring in the size of the data set used, this increase serves as a strong improvement in the random forest model.

VI Conclusion

In conclusion, this investigation explores the significant impact of hyperparameter tuning on the performance of a random forest model in predicting customer churn in a subscription-based telecommunication service. The hyperparameter-tuned model outperformed the baseline model, as well as achieving higher accuracy, recall, precision and f1-score than the model discussed in the literature review. The percentage increases observed in the cross-validation scores imply the extent of improvement achieved through hyperparameter tuning. Although it is small improvements in the performance metrics, it can have a substantial impact on the accuracy and effectiveness of the model in predicting customer churn.

It is important to acknowledge that the impact of hyperparameter tuning may vary depending on factors such as the dataset and specific hyperparameter configurations. Overfitting can occur if hyperparameters are too specific, leading to a reduction in prediction accuracy. In addition, different datasets may have unique features or numerical values that can influence the results ("What are factors for hyperparameter tuning?").

Further research and experimentation with a range of binary classification algorithms is recommended to deepen our understanding of the impact of hyperparameter tuning on performance metrics in predicting customer churn. While the random forest algorithm has been extensively explored in this investigation, considering the broader landscape of algorithms for binary classification will provide helpful insights. By exploring different algorithms and fine-tuning their parameters, we can measure potential improvements across performance metrics and identify the algorithms that display the greatest improvements. Each algorithm has its own unique characteristics and modeling approach, and optimizing their parameters can reveal their full predictive capabilities ("What are some ideas to further improve my investigation? Different algorithms?"). Furthermore, incorporating different datasets with distinct features will enable us to capture a more comprehensive understanding of algorithm performance of hyperparameter tuning. Real-world data is diverse and dynamic, and analyzing its impact on different algorithms will shed light on the practical applications of the findings. The insights gained from this research will not only contribute to advancing the field but also allow telecom companies to implement targeted retention strategies and foster long-term customer relationships.

Overall, the results highlight the importance of hyperparameter tuning in improving the predictive capabilities of the random forest model for customer churn prediction, contributing to the development of effective retention strategies and benefiting the telecommunication industry.

VII Bibliography

Bertsimas, Prof. Dimitris. "4.2 Judge, Jury, and Classifier: An Introduction to Trees." MIT OPENCOURSEWARE. Lecture.

Blastchar. "Telco Customer Churn." *Kaggle*, 16 Feb. 2018, www.kaggle.com/datasets/blastchar/telco-customer-churn?resource=download. Accessed 2 July 2023.

International Business Machines Corporation, editor. "What Is Machine Learning?" *IBM*, www.ibm.com/topics/machine-learning. Accessed 3 July 2023.

---. "What Is Overfitting?" *IBM*, www.ibm.com/topics/overfitting. Accessed 22 July 2023.

---. "What Is Random Forest?" *IBM*, 30 July 2021, www.ibm.com/topics/random-forest. Accessed 12 July 2023.

Jairi, Idriss. "Gini Gain vs Gini Impurity | Decision Tree — a Simple Explanation." *Medium*, 20 Dec. 2021, medium.com/@jairiidriss/gini-gain-vs-gini-impurity-decision-tree-a-simple-explanation-a24ebfeebee9. Accessed 8 July 2023.

"Kaggle." *Wikipedia*, Wikimedia Foundation, 27 Dec. 2023, en.wikipedia.org/wiki/Kaggle. Accessed 9 Jan. 2024.

Koehrsen, Will. "Hyperparameter Tuning the Random Forest in Python." *Medium*, 10 Jan. 2018, towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74. Accessed 14 July 2023.

Kumar, G. Hemanth, et al. *Churn Prediction of Customer in Telecom Industry Using Machine Learning Algorithms*. IJERT, 5 May 2020. *IJERT*, www.ijert.org/research/churn-prediction-of-customer-in-telecom-industry-using-machine-learning-algorithms-IJERTV9IS050022.pdf. Accessed 9 Mar. 2023.

Paddle. "Customer Churn Analysis: One of SaaS' Most Important Processes." *Paddle*, www.paddle.com/resources/customer-churn-analysis. Accessed 21 Dec. 2023.

Pedregosa, Fabian, and Gaël Varoquaux. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research*, 2011, jmlr.csail.mit.edu/papers/v12/pedregosa11a.html. Accessed 26 July 2023.

Sarraju, Vijayalakshmi, et al. *Performance Analysis of Supervised Learning Algorithms on Different Applications*. 12 Nov. 2022. *Semantic Scholar*, https://doi.org/10.5121/csit.2022.121903. Accessed 11 Mar. 2023.

Scornet, Erwan. *ESAIM: Proceedings and Surveys*. 2017. *Google Scholar*, https://doi.org/10.1051/proc/201760144. Accessed 26 July 2023.

Song, Yan-yan, and Ying Lu. *Decision Tree Methods: Applications for Classification and Prediction*. https://doi.org/10.11919/j.issn.1002-0829.215044. Accessed 26 July 2023.

Ullah, Irfan, et al. *A Churn Prediction Model Using Random Forest: Analysis of Machine Learning Techniques for Churn Prediction and Factor Identification in Telecom Sector*. IEEE Access, 6

May 2019. *IEEE Access*, ieeexplore.ieee.org/ielx7/6287639/8600701/08706988.pdf. Accessed 9 Mar. 2023.

Wong, Tzu-Tsung. *Pattern Recognition*. Sept. 2015. *Google Scholar*, https://doi.org/10.1016/j.patcog.2015.03.009. Accessed 27 July 2023.

Ying, Xue. *An Overview of Overfitting and Its Solutions*. Feb. 2019. *IOP Science*, https://doi.org/10.1088/1742-6596/1168/2/022022. Accessed 26 July 2023.

"What are factors for hyperparameter tuning?" prompt. *ChatGPT*, GPT-3.5, OpenAI, 30 November. 2023, chat.openai.com/chat.

"What are some ideas to further improve my investigation? Different algorithms?" prompt. *ChatGPT*, GPT-3.5, OpenAI, 4 February. 2024, chat.openai.com/chat.

VIII Appendix

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score, precision_recall_curve, f1_score
data = pd.read_csv("churndata.csv")
data
data.info()
data = data.drop(['customerID'], axis=1)
print(data.nunique())
data['tenure']=data['tenure'].replace(0,data['tenure'].mean())
data.select_dtypes(include=object).columns
data['TotalCharges'] = pd.to_numeric(data['TotalCharges'], errors='coerce')
data['Churn'] = data['Churn'].map(lambda s :1  if s =='Yes' else 0)
data = pd.get_dummies(data=data, columns=['gender'])
data['Partner'] = data['Partner'].map(lambda s :1  if s =='Yes' else 0)
data['Dependents'] = data['Dependents'].map(lambda s :1  if s =='Yes' else 0)
data['PhoneService'] = data['PhoneService'].map(lambda s :1  if s =='Yes'
else 0)
data['PaperlessBilling'] = data['PaperlessBilling'].map(lambda s :1  if s
=='Yes' else 0)
data['MultipleLines'].replace('No phone service','No', inplace=True)
data['MultipleLines'] = data['MultipleLines'].map(lambda s :1  if s =='Yes'
else 0)
data
data['Has_InternetService'] = data['InternetService'].map(lambda s :0  if s
=='No' else 1)
data['Fiber_optic'] = data['InternetService'].map(lambda s :1  if s =='Fiber
optic' else 0)
data['DSL'] = data['InternetService'].map(lambda s :1  if s =='DSL' else 0)
data.drop(['InternetService'], axis=1, inplace=True)
```

```
data['OnlineSecurity'] = data['OnlineSecurity'].map(lambda s :1  if s =='Yes'
else 0)
data['OnlineBackup'] = data['OnlineBackup'].map(lambda s :1  if s =='Yes'
else 0)
data['DeviceProtection'] = data['DeviceProtection'].map(lambda s :1  if s
=='Yes' else 0)
data['TechSupport'] = data['TechSupport'].map(lambda s :1  if s =='Yes' else
0)
data['StreamingTV'] = data['StreamingTV'].map(lambda s :1  if s =='Yes' else
0)
data['StreamingMovies'] = data['StreamingMovies'].map(lambda s :1  if s
=='Yes' else 0)
data = pd.get_dummies(data=data, columns=['PaymentMethod'])
data = pd.get_dummies(data=data, columns=['Contract'])
data.info()
data.isnull().sum()
data.dropna(inplace=True)
plt.figure(figsize=(35,25))
heatmap = sns.heatmap(data.corr(),annot=True, cmap='coolwarm', fmt=".2f",
linewidths=0.5, annot_kws={"fontsize": 20})
heatmap.set_xticklabels(heatmap.get_xticklabels(), fontsize=20)
heatmap.set_yticklabels(heatmap.get_yticklabels(), fontsize=20)
X = data.drop(['Churn'], axis=1)
y = data['Churn']
data.to_csv('Cleaned_Churn.csv')
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import  cross_val_score
from sklearn.model_selection import RandomizedSearchCV

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=101)

rfclf = RandomForestClassifier(random_state=101)
rfclf.fit(X_train, y_train)
print("Default Random Forest Training Data ")
clf_score = cross_val_score(rfclf, X_train, y_train, cv=5)
print(clf_score)
clf_score.mean()
print("Default Random Forest Testing Data ")
clf_score = cross_val_score(rfclf, X_test, y_test, cv=5)
print(clf_score)
clf_score.mean()
#HYPERPARAM TUNING
param_rand = {
               'criterion': ['gini'],
               'n_estimators' : range(100,1100),
                'max_depth': range(1,10,2),
                'max_features' : ('log2', 'sqrt'),
                'class_weight':[{1: w} for w in [1,1.5]]
                                            }
```

```
Rand_rclf =  RandomizedSearchCV(RandomForestClassifier(random_state=101),
param_rand)
Rand_rclf.fit(X_train, y_train)
print("\nBest parameters \n" + str(Rand_rclf.best_params_))
{'n_estimators': 135, 'max_features': 'log2', 'max_depth': 7, 'criterion':
'gini', 'class_weight': {1: 1}}
rfclf_1 = RandomForestClassifier(random_state = 101, n_estimators=135,
max_features='log2', max_depth=7, criterion='gini', class_weight={1: 1})
rfclf_1.fit(X_train, y_train)
print("Tuned Random Forest Training Data ")
clf_score = cross_val_score(rfclf_1, X_train, y_train, cv=5)
print(clf_score)
clf_score.mean()
print("Tuned Random Forest Testing Data Data ")
clf_score = cross_val_score(rfclf_1, X_test, y_test, cv=5)
print(clf_score)
clf_score.mean()
y_pred = rfclf_1.predict(X_test)
report = classification_report(y_test, y_pred)
print(report)
```