Journey through the Customer Churn Prediction 2020: Lessons, Algorithms, and Future Directions

Introduction

Participating in data science competitions is an exhilarating experience that offers many learning opportunities. Recently, I took part in the Customer Churn Prediction 2020 competition on Kaggle, which challenged participants to predict whether a customer would churn based on historical data. In this report post, I will walk you through my journey, the algorithms I tried, the lessons I learned, and what I would do differently given more time.

Understanding the Problem

Customer churn is a significant issue for businesses, particularly in highly competitive markets like telecommunications, banking, and e-commerce. Identifying customers who are likely to churn allows businesses to take proactive measures to retain them, thereby saving costs and improving customer satisfaction.

The Dataset

The dataset provided for this competition included various features such as account length, number of voice mail messages, international plan, voice mail plan, and more. The goal was to predict the binary target variable churn, indicating whether a customer would leave or not.

Data Preprocessing

Initial Exploratory Data Analysis (EDA)

1. **Data Cleaning:** Checked for and handled missing values.
2. **Data Encoding:** Used frequency encoding for categorical variables like state, and one-hot encoding for area_code.
3. **Normalization:** Applied MinMaxScaler to normalize the numerical features.

Outlier Detection

Outliers can significantly skew the results of machine learning models. I used Z-score to identify and visualize outliers in features like account_length and number_vmail_messages.

Algorithms Tried

1. Naive Bayes

- **Why:** Naive Bayes is simple yet powerful for binary classification problems. It provides a good baseline model.
- **Result:** The performance was decent but not competitive enough.

2. Decision Tree Classifier

- **Why:** Decision Trees are interpretable and can handle both numerical and categorical data without the need for extensive preprocessing.
- **Result:** Achieved reasonable accuracy but prone to overfitting.

3. Random Forest Classifier

- **Why:** Random Forest is an ensemble method that reduces overfitting by averaging multiple decision trees.
- **Result:** Showed better performance and reduced overfitting compared to a single Decision Tree.

4. Support Vector Machine (SVM)

- **Why:** SVMs are effective in high-dimensional spaces and can create complex decision boundaries.
- **Result:** The performance was satisfactory but computationally expensive for large datasets.

5. k-Nearest Neighbors (k-NN)

- **Why:** k-NN is simple and intuitive, making predictions based on the majority class of nearest neighbors.
- **Result:** Performed well but struggled with large datasets and imbalanced classes.

6. Gradient Boosting Classifier

- **Why:** Gradient Boosting is an ensemble technique that builds models sequentially, with each model correcting errors of the previous one.

- **Result:** This model provided the best results due to its ability to handle complex patterns in the data.

Hyperparameter Tuning

For each algorithm, I performed hyperparameter tuning using GridSearchCV to find the optimal parameters that maximize the model performance. This step was crucial in improving the accuracy and robustness of the models.

Model Evaluation

I used 10-fold cross-validation to evaluate the models. This technique ensures that the model performance is not dependent on a specific train-test split and provides a better estimate of the model's generalization capability.

Results

The following table summarizes the performance of each model:

Model	Accuracy (10-fold CV)
Naive Bayes	84.5%
Decision Tree	85.7%
Random Forest	87.9%
SVM	86.3%
k-NN	85.2%
Gradient Boosting	88.6%

Gradient Boosting emerged as the best model with an accuracy of 88.6%.

Lessons Learned

1. **Importance of Data Preprocessing:** Proper handling of missing values, encoding categorical variables, and normalizing numerical features are critical for the performance of machine learning models.
2. **Model Selection:** Ensemble methods like Random Forest and Gradient Boosting outperform simpler models due to their ability to reduce overfitting and handle complex patterns.

3. **Hyperparameter Tuning:** Tuning hyperparameters significantly improves model performance. It is worth investing time in this step.
4. **Cross-Validation:** Ensures that the model performance is reliable and not dependent on a specific train-test split.

What Could Have Been Done Differently

1. **Feature Engineering:** Given more time, I would have focused more on feature engineering to create new features that could provide additional predictive power.
2. **Handling Imbalanced Data:** Implementing techniques such as SMOTE (Synthetic Minority Over-sampling Technique) to handle class imbalance could have improved the model's ability to predict the minority class (churn).
3. **Advanced Algorithms:** Exploring more advanced algorithms like XGBoost or LightGBM, which are known for their superior performance in many Kaggle competitions, could have provided an edge.
4. **Model Stacking:** Combining predictions from multiple models (model stacking) might have further improved the accuracy.

Conclusion

Participating in the Customer Churn Prediction 2020 competition was a highly educational experience. I explored various algorithms, learned the importance of data preprocessing, and realized the significance of hyperparameter tuning. While Gradient Boosting provided the best results, there is always room for improvement, especially with feature engineering and advanced modeling techniques.