Final Year Interim Report

Full Unit – Interim Report

Customer Churn Prediction: A Comparative Analysis of

Machine Learning Algorithms

Abdulahi Said

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Dr Anand Subramoney

Department of Computer Science

Royal Holloway, University of London

December 13, 2024

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name:

Date of Submission:

Signature:

VIDEO LINK: https://youtu.be/PqTXaV_6Gjg

Table of Contents

Abstract

When clients discontinue using a service, customer churn becomes a critical issue across many industries, especially for subscription-based businesses. In such sectors, customer satisfaction and retention are vital to ensuring steady revenue streams. With intense competition, every lost customer can represent a direct gain for competitors [1]. On average, businesses lose between 10% and 25% of their customer base annually due to churn, with some industries, such as wholesale, experiencing rates as high as 56% [2][3].

High churn rates undermine a business's ability to establish and maintain a loyal customer base, essential for long-term profitability. Studies have shown that retaining customers is much more profitable and sustainable [5]. This is because customer retention requires fewer resources compared to acquiring new customers and because loyal customers tend to spend more over time. As a result, businesses that focus on strengthening relationships with existing clients can boost their profitability while ensuring steady growth.

To address this known issue, this project aims to conduct a comparative analysis of various machine learning (**ML**) algorithms designed to predict customer churn effectively. These algorithms include K-Nearest Neighbours (**KNN**), Decision Trees (**DT**), Support Vector Machines (**SVM**), and Random Forest **(RF**). A significant challenge in churn prediction is class imbalance. This occurs as the number of churners is outnumbered by customers that remain. This can lead to biased model predictions, increasing the number of false positives and false negatives. Such inaccuracies have damaging repercussions on businesses as they may waste financial and marketing resources targeting the wrong customers while overlooking actual customers who would leave. To deal with these issues, I will implement robust data preprocessing techniques, such as handling missing values and scaling features. Additionally, I will explore ensemble methods to improve model accuracy and generalisation [6].

Furthermore, my project will apply these ML models to a benchmark dataset. Each algorithm's performance will be measured using various measures, including precision, recall, F1-score, and Area Under the Receiver Operating Characteristic Curve (**AUC-ROC**), thus ensuring comprehensive model assessment. To enhance accessibility for non-technical users, I will develop a GUI that allows users to visualise the performance of selected ML models, making it more user-friendly and intuitive.

Project Specification

Aims: To implement and compare on benchmark data sets various machine learning algorithms

Background: Machine learning allows us to write computer programs to solve many complex problems: instead of solving a problem directly, the program can learn to solve a class of problems given a training set produced by a teacher. This project will involve implementing a range of machine learning algorithms, from the simplest to sophisticated, and studying their empirical performance using methods such as cross-validation and ROC analysis. In this project you will learn valuable skills prized by employers using data mining.

Early Deliverables

1. You will implement simple machine learning algorithms such as nearest neighbours and decision trees.
2. They will be tested using simple artificial data sets.
3. Report: a description of 1-nearest neighbour and k-nearest neighbours algorithms, with different strategies for breaking the ties;
4. Report: a description of decision trees using different measures of uniformity.

Final Deliverables

1. May include nearest neighbours using kernels and multi-class support vector machines.
2. The algorithms will be studied empirically on benchmark data sets such as those available from the UCI data repository and the Delve repository.
3. For many of these data set judicious preprocessing (including normalisation of attributes or examples) will be essential.
4. The performance of all algorithms will be explored using a hold-out test set and cross-validation.
5. The overall program will have a full object-oriented design, with a full implementation life cycle using modern software engineering principles.
6. Ideally, it will have a graphical user interface.
7. The report will describe: the theory behind the algorithms.
8. The report will describe: the implementation issues necessary to apply the theory.
9. The report will describe: the software engineering process involved in generating your software.
10. The report will describe: computational experiments with different data sets and parameters.

Suggested Extensions

- Modifications of known algorithms.
- Dealing with machine learning problems with asymmetrical errors (such as spam detection) and ROC analysis.
- Nontrivial adaptations of known algorithms to applications in a specific area, such as medical diagnosis or option pricing.
- Exploring the cross-validation procedure, in particular the leave-one out procedure. What is the optional number of folds?
- Comparative study of different strategies of reducing multi-class classifiers to binary classifiers (such as one-against-one, one-against-the-rest, coding-based).

Reading

- Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning. Second edition. Springer, New York, 2009.
- Tom M. Mitchell. Machine Learning. McGrow-Hill, New York, 1997.
- Vladimir N. Vapnik. The Nature of Statistical Learning Theory. Second edition. Springer, New York, 2000.
- Vladimir N. Vapnik. Statistical Learning Theory. Wiley, New York, 1998.
- Seth Hettich and Steven D. Bay. The UCI KDD Archive. University of California, Department of Information and Computer Science, Irvine, CA, 1999,

Chapter 1: **Background and Key Concepts**

1.1 What is Machine Learning?

Machine Learning is a subset of artificial intelligence driven by algorithms designed to identify patterns in data. These patterns allow the algorithm to make predictions without requiring explicit programming for each task, whether it is predicting if an image is a cat or predicting stock prices. The adaptability of machine learning makes it a powerful tool in data-driven applications, helping to tackle many challenges across various industries.

Machine Learning can be classified into three categories:

1. **Supervised Learning**: Uses labelled data that contains a feature and target to learn patterns and relationships. Once trained, the model can generalise from the learned patterns to predict the target labels of unseen data. For instance, predicting churn is a supervised problem, where historical data on whether customers have churned or not is used to train the model to make accurate predictions.

2. **Unsupervised Learning:** Utilises data that doesn't have preexisting features and label mappings to discover patterns and relationships. An example of usage would be clustering customers based on similar behaviours.

3. **Reinforcement Learning** is quite different from supervised and unsupervised learning. It involves learning through trial and error by interacting with an environment to achieve the most optimal results through sequential decision-making and actions.

Given that my project is customer churn prediction, I will implement supervised learning using features such as tenure, monthly, charges, services and payment methods to predict whether a customer would churn. This would then allow businesses to take proactive measures for customers retention.

1.2 Classification vs Regression

Depending on the predicted label, most machine-learning problems can be classified into regression or classification tasks. Regression is a type of supervised learning that predicts continuous numerical values, such as the temperature or house prices. On the other hand, classification predicts using discrete categories or labels such as 'yes' and 'no'.

Customer churn prediction would be classified as a classification problem. This is because the goal is to assign the two discrete labels churned (1) and not churned (0). It is not a regression problem, as it does not involve predicting a continuous value representing the likelihood of churn. Instead, it focuses on categorising customers based on their likelihood of discontinuing the service, making classification the appropriate approach for this project.

1.3 Binary vs. Multi-Class Classification

In machine learning, classification problems can be categorised into binary and multi-class. Binary classification is where only two possible class labels can be assigned, such as ‘churned’ and ‘not churned’. Multi-class classification involves predicting the label of a data point from three or more possible categories. An extension of the churn example for multi-class classification could be adding different likelihood risks of churn such as “none”, “low”,’ medium’ and “high”.

In my project, the churn prediction would be initially treated as a binary classification problem, where the goal is to categorise each customer into one of two categories: those likely to churn and those not likely to churn. This approach simplifies the problem, providing a good foundation for me to carry out experimentation and analysis. There is also the flexibility to later expand into a multi-class classification by adding different levels of churn risk.

1.4 Dataset I will be using

The Dataset that I have chosen to use is a Telecoms customer churn dataset from IMB[7]. This dataset contains 7043 customers with 21 features capturing services used by customers and subscription details.

1.4.1 Key Features:

- **Customer Demographics**: Information such as gender , senior citizen status and whether they have dependants.
- **Service Details**: Features like whether the customer has phone service, multiple lines, or internet service and whether the customer uses other services such as security and streaming services.
- **Contract Information**: The type of contract (month-to-month, one-year, two-year), payment method, and whether paperless billing is enabled.
- **Charges and Tenure**:
- **Tenure**: The number of months a customer has been with the telecoms service.
- **Monthly Charges**: The monthly cost of the services used
- **Total Charges**: The cumulative charges over the entire tenure.

1.4.2 Label

The label in this dataset would be churn, a binary label indicating whether a customer has churned “yes” or hasn’t churned “no”. I have converted these into 0 and 1, 0 representing not churned and 1 representing churn.

1.4.3 Why this dataset?

I selected this dataset because it was highly rated on Kaggle for its quality and because IBM, a well-known leader in telecommunications services, produced it. Since it simulates real-world churn scenarios, it makes it a suitable choice for machine-learning models.

Chapter 2: **Performance Evaluation**

2.1 Introduction of Model Evaluators

Evaluating models is a crucial part of this project. The primary objective is to compare a range of machine learning algorithms to determine the most effective for predicting churn. It is also key as it informs whether the ML model generalises well to unseen data and the model's capabilities to classify whether a customer is a churner correctly.

As I have established before, this is a classification problem, and the choice of metrics is important. We cannot use mean squared error (MSE) or R-squared as the label is not continuous. Metrics such as precision, accuracy, recall and F1-score are more suitable for classification and give a detailed insight into the model's performance. For example, a high recall score ensures that most actual churners are identified, which is key if companies want to target those customers with retention strategies. Similarly, a high precision score minimises the risk of misclassifying non-churners as churners, avoiding unnecessary usage of resources and intervention to convince customers to stay.

2.2 Evaluation Metrics explained

- **True Positive (TP)**: A churner correctly identified as a churner.

- **False Positive (FP)**: A non-churner incorrectly identified as a churner.

- **False Negative (FN)**: A churner incorrectly identified as a non-churner.

- **True Negative (TN)**: A non-churner correctly identified as a non-churner.

2.2.1 Accuracy

The most common metric is accuracy; it measures a model's number of correct predictions divided by the total number of predictions. While this metric is simple to use, it can be misleading as it does not account for the number of false negatives and false positives. Since the churn dataset is imbalanced, where the majority of customers do not churn, as compared to churning, the model could have a high accuracy score based on predicting the majority class, rendering this metric less useful in identifying actual churners(True negative).

2.2.2 Precision

Precision is the proportion of true positives identified by the model out of all instances it predicted as positive. In relation to our churn prediction, precision measures the number of correctly identified churners (true positives) divided by the total customers predicted as churners (true positives + false positives).
Precision = (True Positives) / (True Positives + False Positives)

A high precision score indicates that the model minimises false positives, ensuring resources are focused on actual churners rather than misclassified non-churners.

2.2.3 Recall

Recall, also known as sensitivity, measures a model's ability to identify the number of true positives in a dataset. In relation to our churn prediction, this would be the proportion of correctly identified churners out of all instances predicted as churners.

Recall = (True Positives) / (False Negatives + True Positives)

The recall score must be high so that at risk customers can be identified and targeted with retention strategies. A low recall score means that many actual churners are missed (false negative), causing lost revenue as these customers would leave without any intervention.

2.2.4 F1-Score

A metric that combines recall and precision, which is used in classification problems. It provides a balanced assessment of a model's performance as it considers both recall and precision. It is very useful when datasets are imbalanced, which they are in this case.

F1-Score = 2 × (Precision * Recall) / (Precision + Recall)

A high F1 score shows that the model effectively identifies churners and keeps false positives low. This is important since, in a business environment, misclassification can affect financial and resource consequences.

2.2.5 F2 -Score

A variation of the F1-score but places a greater emphasis on recall than precision. It’s useful when identifying actual churners is more important than avoiding false positives.

- β2 = 2
- F2-Score = (1+β2) × ((Precision × Recall) / ((β2×Precision) + Recall))

F2-Score has limitations so I would use it in conjunction with all the other metrics

Chapter 3: **Explorative Data Analysis and Data Preprocessing**

Explorative Data Analysis (EDA) and data preprocessing are critical steps in the machine learning pipeline. Explorative data analysis provides a deeper understanding of the dataset by visualising patterns through graphs and statistical summaries, detecting anomalies. The deeper understanding gained from EDA aids in parameter tuning and feature selection, guiding our overall modelling approach and results [8].

Data preprocessing essentially transforms the raw data into clean data by handling missing data values and encoding data in a way that is suitable for the ML model and normalising features. These steps ensure that the model is trained on quality and reliable data, thus enhancing the accuracy and reliability of our results [9].

3.1 Explorative Data Analysis on Churn Dataset

The dataset I used in this project, described in section 2.4, is the Telco Customer Churn dataset. This section focuses on the insights I gained from carrying out exploratory data analysis.

To gain an initial understanding of my data, I converted it into a data frame and used the command .head(). This provided an overview of the structure, including feature names, data types, and sample values, ensuring familiarity with the dataset's layout. Since missing data is a critical aspect, I utilised a heatmap to output which attributes have missing data as shown in figure1.

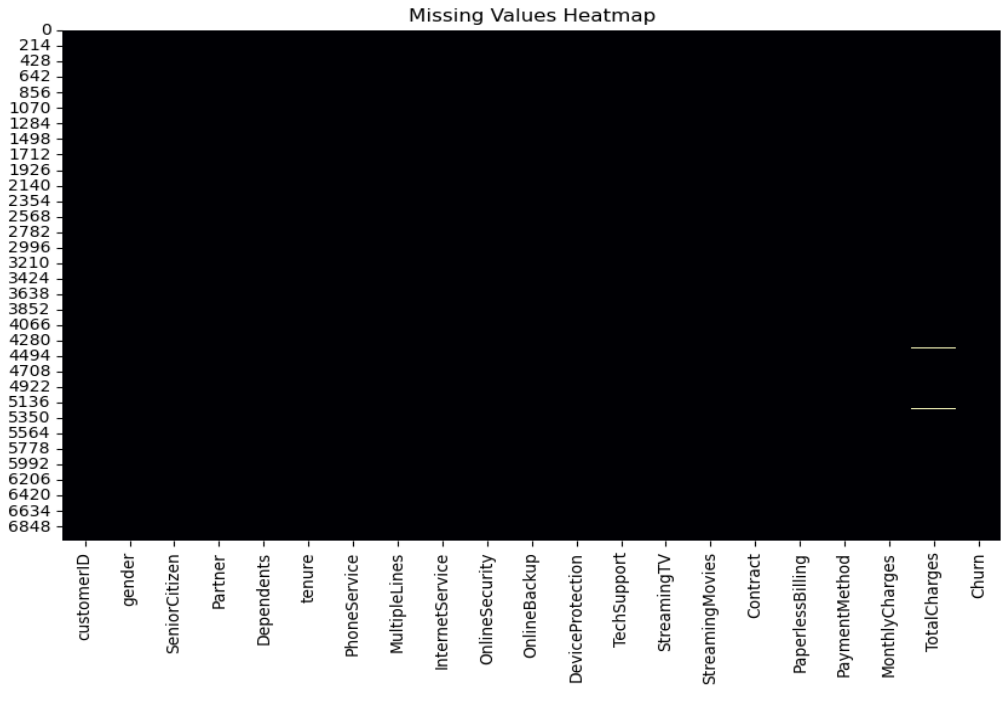

Figure 1. Heatmap of missing Values in the Dataset

The heatmap revealed that total charges had missing data, and upon closer examination, eleven customers had missing data for the Total charges column. This insight was essential for data preprocessing phase.

3.1.1 Insights from Visualising Relationship

I began investigating the relationships between different features through various visual plots that include, histograms, boxplots and bar charts.

1. Payment Methods and Churn: The bar charts showed that customers using electronic checks have a higher churn rate than customers using other methods, such as credit cards and bank transfers. This could imply customer dissatisfaction with certain payment methods.

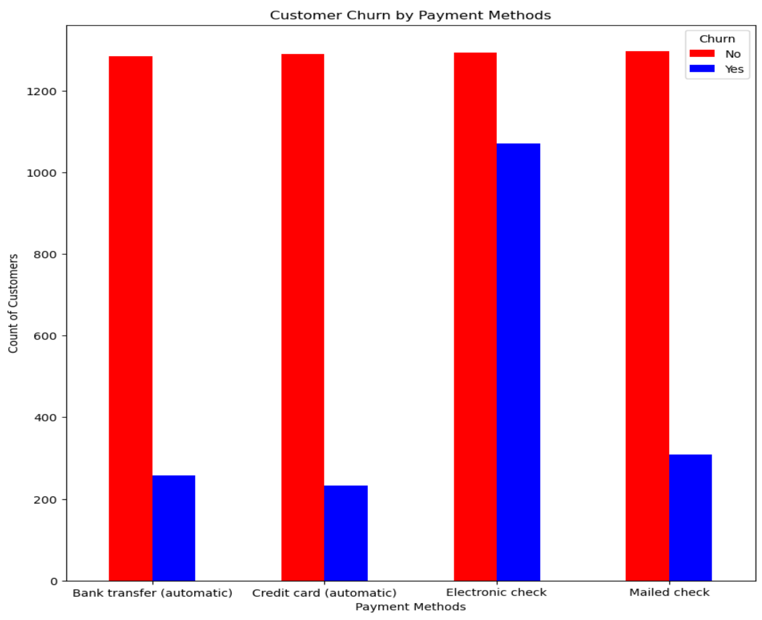

Figure 2. Customer Churn by payments Bar Chart

2. **Impact of Monthly Charges**: A boxplot of Monthly Charges against Churn revealed customers with higher monthly charges show a greater likelihood of churning, indicated by the higher median and narrower spread of charges for churners compared to non-churners in figure 3. This suggests highly to me that monthly charge is a key feature.

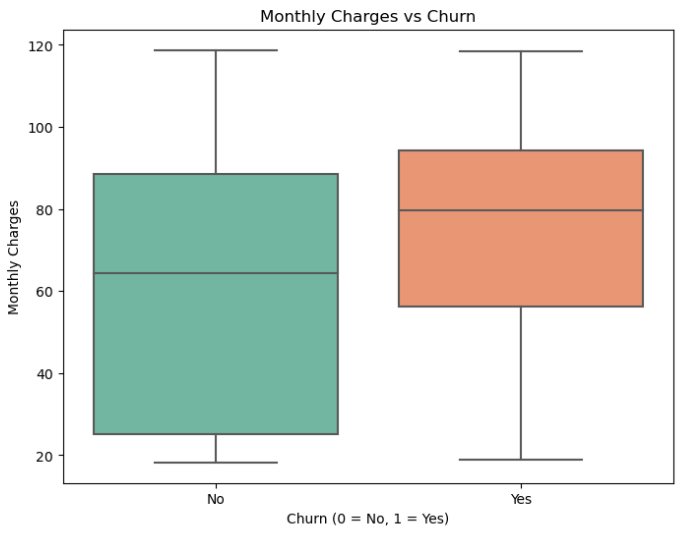

Figure 3. A box plot of Monthly Charges vs Churn

Figure 4.

3. **Correlation Matrix:** Figure 4 displays a correlation matrix of the numerical features of the dataset. From Figure 4 we can infer a strong positive correlation between Total Charges and Tenure. Also, monthly charges and tenure show a weak positive correlation (0.247), while senior citizens have a minimal correlation with other features, indicating limited predictive benefits for churn.

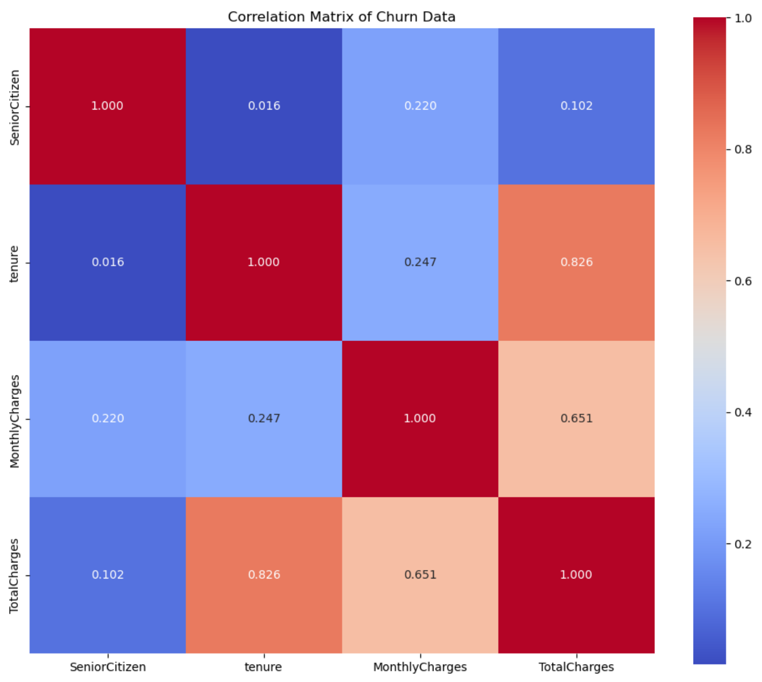

Figure 5. Correlation Matrix of Churn Dataset

3.2 Data Pre-Processing

3.2.1 Handling Missing Values

During my exploratory data analysis, I discovered that the Total Charges column had 11 missing values. My initial decision was to delete these 11 rows from the dataset, given that all those customers haven't churned and also given the class imbalance. I deemed it would have been fine. However, upon closer inspection, I realised that these rows included valid Monthly Charges and Tenure values, which could be used to manually calculate the missing Total Charges values. Since Tenure represents the number of months a customer has been with the service provider, and Monthly Charges reflect the recurring monthly cost, the missing Total Charges values were calculated using the formula: Total Charges = Monthly Charges * Tenure. By utilising this relationship this ensured that no data was wasted and have maintained the integrity of the dataset.

Chapter 4: **K-Nearest Neighbours (KNN) model**

4.1 Background Theory

The K-Nearest Neighbour (KNN) algorithm is a supervised learning method widely used for classification tasks due to its simplicity and effectiveness. It utilises the nearest neighbour of the given data point to determine the class. The parameter K represents the number of neighbours considered, and the final prediction would be based on a majority vote. In the case of a tie, various methods can be used to resolve it, such as random voting. However, after running KNN extensively on the dataset utilising a range of k from 1-50, I encountered 0 ties, and therefore, I did not investigate this issue extensively. However, for precaution in the event of a tie, the first class among the tied options is selected.

Furthermore, KNN is a lazy learner, and no real training is required, as distance computations between data points are carried out during the prediction phase. This makes KNN a memory-intensive algorithm, especially with large, high-dimensional datasets [10].

4.1.1 How are distances between datapoints calculated?

In KNN, the classification of an unknown data point heavily depends on the distance between labelled data points and the selected k parameter. This makes distance a critical step, but how do we compute the distances? There are several distance metrics, each suited to different purposes:

1. Euclidean distance:

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

Figure 6. Euclidean distance equation

This equation calculates the straight line distance between 2 data points by summing the difference between the two coordinates and taking the square root.

2. Manhattan Distance: It computes the sum of the absolute difference between the coordinates of the data points.

   d = Σ | p_i – q_i |

   P and Q are vectors

   i represents the ith element of the vector

   Manhattan distance is generally used on gird paths

3. Cosine Similarity measures the angular difference between data points and is useful when the dataset's dimensionality is high, although it's not an actual distance metric.

   Cosine Similarity = (x · y) / (||x|| × ||y||)

4.2 Implementation of KNN

I decided to implement KNN from scratch without using libraries such as SKLearn. The main goal was to test my understanding of the theory and see how well it performs in customer churn classification. After brainstorming, my initial implementation was to create a KNN class that encapsulates the algorithm's key functionalities.

The initialisation function handled the K parameter and utilised safeguards to prevent erroneous entries, such as negative numbers. If not specified, neighbours were defaulted to 1. Since there is no real training, the fit method stores the corresponding training data: the features (X_train) and the labels (y_train). The next stage was determining the k-nearest neighbours for a given data point and selecting an appropriate distance metric. I chose the **Euclidean distance metric**, which is widely used and suitable for numerical data. It calculates the straight-line distance between two points in the feature space, defined in Figure 6.

To optimise the distance computations, I utilised NumPy's vectorised operations, significantly improving calculation speed compared to looping through the entire dataset. This approach ensured efficient processing, even with larger datasets.

The code for calculating Euclidean distance is shown below:

```
    def euclidean_distance(self,p1,p2):
        return np.sqrt(np.sum((np.array(p1) - np.array(p2)) ** 2))
```

When the KNN model does prediction, the algorithm calculates the distance between the test data point and every data point in the training set. This would involve iterating through the training dataset and computing the Euclidean distance for each pair. The issue with this is that sorting becomes timely, especially since my dataset is over 7000. I initially used a general insertion sort function I created but quickly realised that this method was inefficient as the time complexity of insertion sort is **$O(n^2)$,** meaning that, in the worst case, the time taken to sort a list is proportional to the square of the number of elements in the dataset. To address this issue, I utilised python’s heapq library, which efficiently identifies k smallest distances without sorting the whole list. This optimisation significantly reduced the run time of my KNN model.

Code behind getting K-nearest neighbour:

```
def get_nearest_neighbour(self, test_sample):
    # Checks if we set K > size of sample and deals with it accordingly
    k = min(self.n_neighbours, len(self.X_train))

    # Calculate all distances without sorting
    distances = []
    for train_sample, label in zip(self.X_train, self.y_train):
        distance = self.euclidean_distance(train_sample, test_sample)
        distances.append((distance, label))

    # Use heapq to get the k smallest distances directly
    k_nearest_neighbours  =  heapq.nsmallest(k,  distances,  key=lambda  x:
x[0])
    return k_nearest_neighbours
```

4.3 Running my KNN model

After implementing the KNN model and testing it utilising TDD development and unit testing, I then ran it on my churn dataset. To evaluate the model results I utilised my evaluator class that outputs accuracy, precision, recall, F1-score and F2-score. The aim was to find the optimal value of k, ensuring the best possible results.

4.3.1 My setup for investigating k-values

To evaluate the model, I split the data into test and train, with 20% of the data being for testing and the other 80% for training. I then created a function to test k from range 1-50, storing results in an array, which would then be plotted as a line graph. This would make investigating the optimal K value much easier.

4.3.2 Model Results across different K-values:

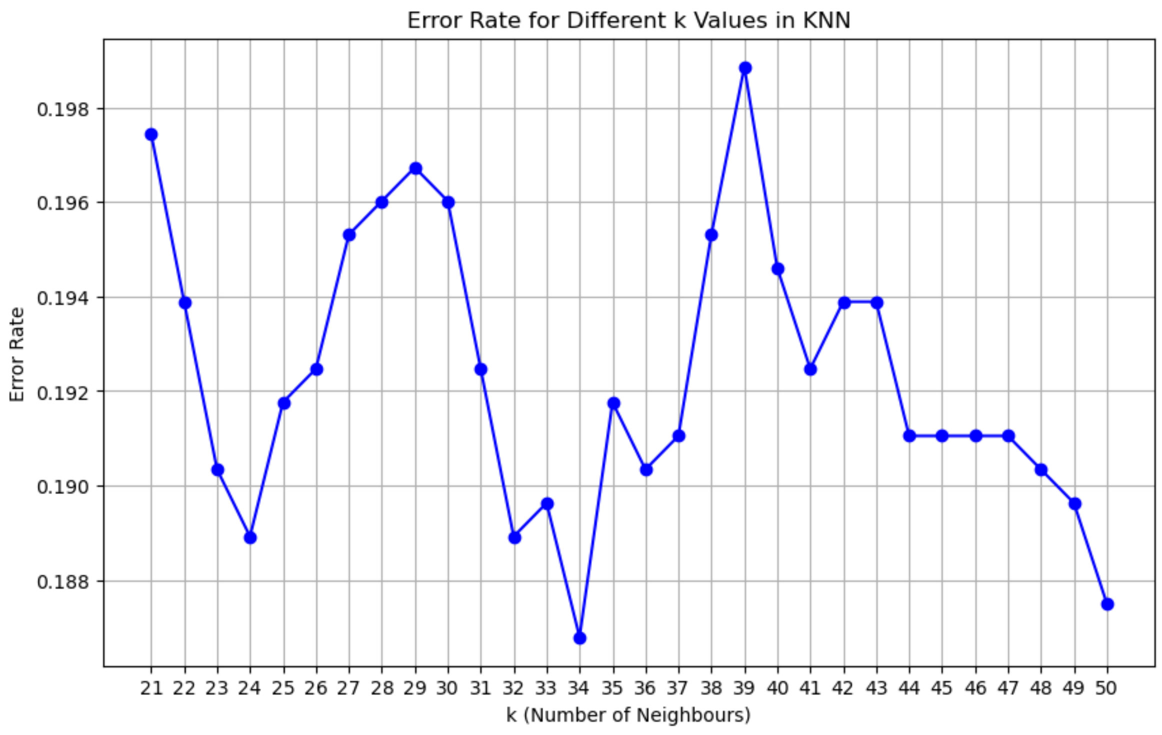

Figure 7. Error rate for different K-values of KNN model

While you cannot see from graph the error rates of k from range 1-20, the error rate was higher than range k 21-50 . K values in the range 1-20 exhibited the highest error rates but were gradually declining as k increased, and then there was a noticeable drop at k = 21. Its likely that very small values of k performed poorly as they are much more likely to capture noise and any outliers in the dataset that would lead to more unreliable predictions.

For k values greater than 20, the error rate appeared to fluctuate significantly in the graph; however, these fluctuations were within very small margins, in the range of 3 decimal places. This is why the graph looks very volatile, but in reality, it is stable. Looking at the graph, the optimal k was when k = 34, as it had the lowest error rate. For K values greater than 34 it continued to fluctuate.

4.3.3 Optimal K value comparison using different Normalisation.

After identifying the optimal k-value as k=34k, I decided to investigate the impact of different normalisation techniques on the model's performance. Normalisation is essential for distance-based algorithms like KNN, ensuring that features with larger values do not disproportionately influence the distance computation [10].
For this comparison, I used two widely adopted techniques: Min-Max Scaling and Z-Score Scaling**.**

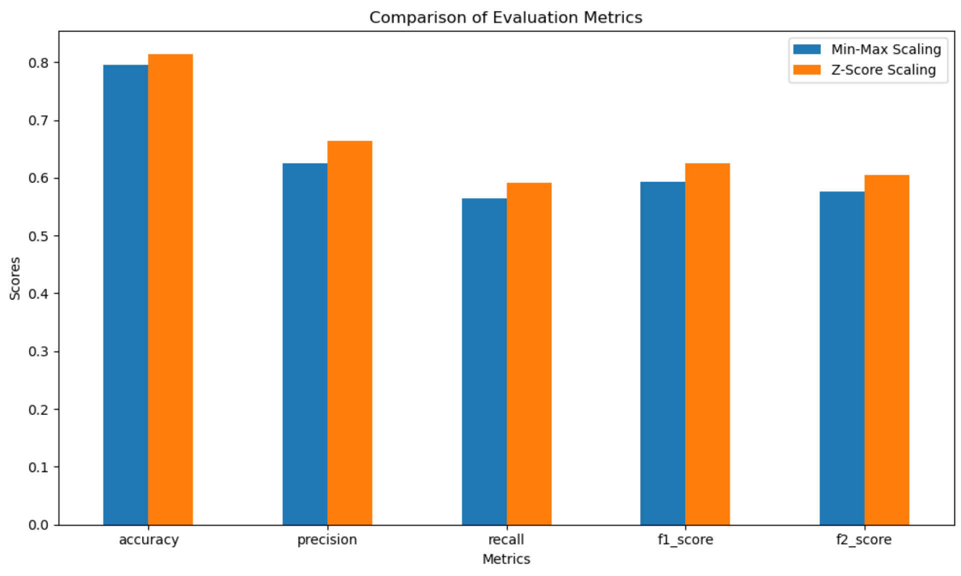

Figure 8. K=34, knn model with different Normalisation method

The performance of k = 34 was assessed using my evaluator metric class, and Figure 8 displays the performance between the two normalisation methods.

Observations:

- Accuracy: Both normalisation methods had a fairly high accuracy, but z-score performed slightly better than min max.
- Precision and recall: Z-score consistently achieved higher precision and recall than min max. Thus indicating it manages the feature distribution of the churn dataset more effectively
- F1/F2 Score: z-score demonstrated a slightly higher score than min-max normalisation.

This investigation revealed that normalisation techniques affect KNN model performance, but not significantly.

Despite these variations, the KNN model demonstrated decent overall performance. These results for precision, recall, and f1-score are understandable given the imbalanced nature of the dataset and the inherent characteristics of KNN as a lazy learner.

Chapter 5: **Decision Tree Model**

5.1 Background Theory

5.1.1 What is a decision tree?

A decision tree is a supervised learning algorithm that can be used for both classification and regression. Structurally, it resembles a tree structure consisting of a root node, which is the tree's base, and branching from it is a hierarchy of decision nodes terminated by leaf nodes. Leaf nodes are nodes with no children.

A decision tree is a supervised learning algorithm that can be used for both classification and regression. Structurally, it resembles a tree consisting of a root node, which is the tree's base. Branching from it is a hierarchy of decision nodes terminated by leaf nodes. Leaf nodes have no further partitions, meaning they have no children. Also, each decision node represents a splitting point where the data is divided into subsets based on a feature value or condition. The criteria for splitting could be vast, from whether monthly charges exceed a certain number to categorical questions such as whether the customers have a dependant. This branching process would continue until partitions of the data are pure or another stopping criterion is met.

Purity refers to how homogeneous the data is after splitting. A data partitioning is considered pure if and only if all the data points belong to that class. For example, in my churn prediction, a subset that contains data points of those with higher monthly charges than 27 could be considered pure. Decision trees aim to decide the splitting of data by maximising purity.

5.1.2 Different Measures of purity

Many different measures of purity decide splits and I will be focusing on three:

1. **Gini Impurity**: returns the likelihood of new random data being misclassified if given a random label from the possible classes. Its result Ranges from 0-0.5

   0 value indicates a perfect purity (each datapoint belongs to single class) and 0.5 means maximum impurity (the datapoints are split equally between the classes).

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Figure 9. Gini impurity equation

2. **Entropy :**

   Pi is the proportion of data points, in this case, customers that belong to the same class.

$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

   0 Indicates a completely pure subset (all customers belong to the same class i.e churners or not churners

Figure 10.

3. **Classification error:** It measures the proportion of incorrectly classified data points in a subset. A very simple metric and equation is: Classification Error =1 − max(pi).

5.2 Implementation of Decision Trees

Like my KNN model, I implemented it from scratch and used the background theory I learnt to develop it. My initial implementation was to have a single class that would handle the basic tree construction with different uniformity measures and then predict unseen data. However, looking at future models to be developed and extensions of the decision trees model, such as implementing ensemble methods like random forests and gradient-boosted Boosted Trees. I decided to opt for a more modular approach by developing two classes: Node and Decision Tree. There were some challenges, but it was worth it for the flexibility this class would provide for later usage.

Why only two classes? The **Node** class is designed to represent the building blocks of the decision tree.

```
class Node:
       def   __init__(self,   feature=None,   threshold=None,   left=None,
right=None, *, value=None):

        """
        Initialises a node in the decision tree.

        Parameters:
        - feature: Index of the feature used for splitting.
        - threshold: Threshold value for the split.
        - left: Left child node.
        - right: Right child node.
        - value: Class label if the node is a leaf.
        """
        self.feature = feature
        self.threshold = threshold
        self.left = left
        self.right = right
        self.value = value

    def is_leaf_node(self):
        return self.value is not None
```

This initialisation function contains the essential attributes and has a leaf function to determine if the node has no children, thus simplifying tree traversal.
Developing this node class allowed it to be reusable and flexible. It also made it much easier to test and debug when the tree-building process occurred.

5.2.1 Decision Tree class

Having developed the node class, I then moved on to developing the main class(decision tree).The decision tree class's purpose is to construct the tree, train it using the churn dataset, and make predictions. The fit method was the starting point of the tree construction and would carry out a recursive call to the grow function, which I will discuss.

Challenges and notable methods for decision tree class:

1. Handling the splitting criteria: Initially, I only implemented gini impurity as the default for deciding when to split. However, I realised that if I were to investigate how different uniformity measures affect decision trees, I would have to make them modular. To handle this, I created a uniformity measure parameter that would take 3 of the different ones, as discussed, and developed a flexible impurity function to calculate impurity based on the chosen metric.

   Code for impurity, utilising the equations for different uniformity measurements:

   ```
    def impurity(self, y):
   """
   Calculate node impurity based on the chosen uniformity_measure.
   """
   proportions = np.bincount(y) / len(y) # p(X)

   if self.uniformity_measure == "gini":
       return 1 - np.sum(proportions ** 2)
   elif self.uniformity_measure == "entropy":
       return -np.sum([p * np.log2(p) for p in proportions if p> 0])
   elif self.uniformity_measure == "error":
       return 1 - np.max(proportions)
   ```

2. Deciding on the stopping criteria and how to manage it was an issue. In my first basic implementation, the tree would grow excessively, leading to overfitting, especially since I have about 30+ features after encoding. This would lead to overfitting, and to resolve this, I added parameters such as max_depth and min_samples_split; this would allow me to control the complexity of the tree. This would also allow me to carry out parameter tuning to allow me to get the best performance results from decision trees model.

3. Traversing the tree is key in the decision tree because to predict new data points, I would need to traverse the tree from the root node to the leaf node. There are several tree traversal methods, such as postorder, inorder and preorder, but for predictions of decision trees, the traversal is very specific. The feature values and thresholds guide the traversal in decision trees as each node, the feature value of the datapoint, would be compared against the threshold value. I recursively checked the feature values against thresholds and moved left and right in the tree until a leaf node was reached. The traversal was made more straightforward due to the design of my node class.

4. To ensure the model's robustness, I implemented verification and edge case handling. This includes addressing scenarios where empty splits are selected, causing errors and added complexity. Furthermore, implementing practical functions such as saving and loading models for my customer churn project was beneficial. By utilising Python's Pickle library, the trained model could be reused and computational resources saved.

5.3 Results and Running of my Decision tree model

I ran my decision tree on the churn dataset and saw its results. Before running, I split the data using 30% for testing and 70% for training. I plan to use cross-validation to improve the validity of my results in the future.

Gini Impurity

- Performance Metrics: Accuracy: 0.793 (79.3%), Precision: 0.620 (62.0%), Recall: 0.618 (61.8%), F1 Score: 0.619 (61.9%), F2 Score: 0.619 (61.9%)

Entropy

- Performance metrics: Accuracy: 0.789 (78.9%), Precision: 0.616 (61.6%), Recall: 0.592 (59.2%), F1 Score: 0.604 (60.4%), F2 Score: 0.597 (59.7%).

Classification error

- Performance Metrics: Accuracy: 0.773 (77.3%), Precision: 0.677 (67.7%), Recall: 0.317 (31.7%), F1 Score: 0.432 (43.2%), F2 Score: 0.355 (35.5%).

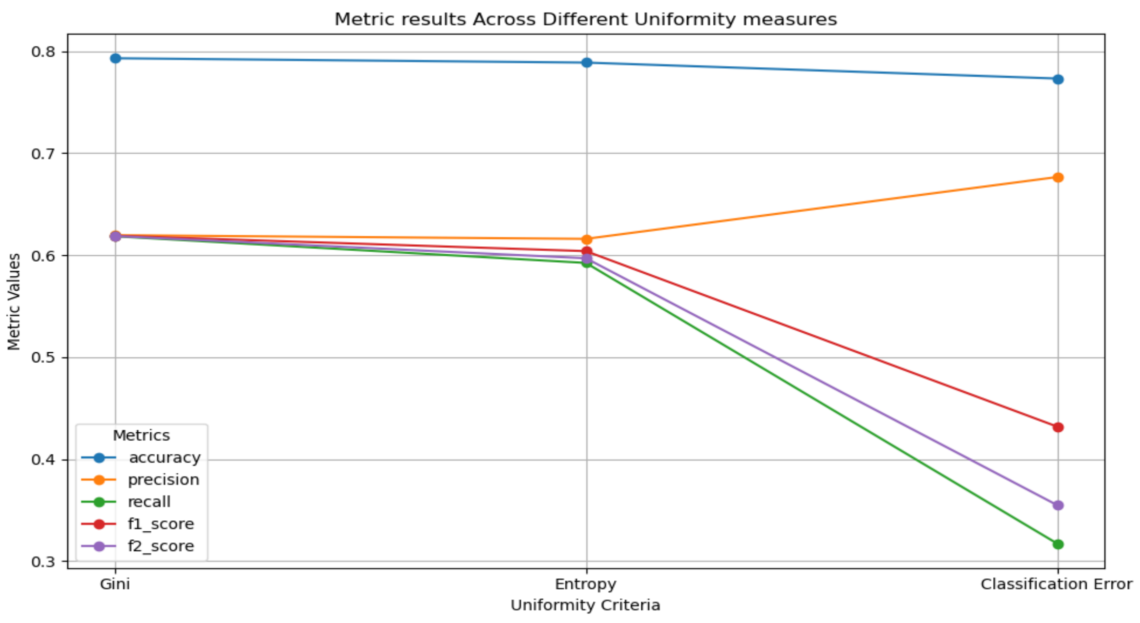

Figure 11. Results for different uniformity measures.

5.3.1 Analysis of results:

The Gini Impurity achieved the highest overall performance, demonstrating its effectiveness at identifying true positives and avoiding false positives.

Gini Impurity was the best split criterion, providing the best balance between the two classes, with an accuracy of 79.3%, precision of 62.0%, and recall of 61.8%, making it the most reliable for this dataset. Entropy performed slightly worse, with an accuracy of 78.9% and a lower recall of 59.2%, leading to slightly lower F1 and F2 scores. Classification Error performed the worst, with a very low recall of 31.7%, an accuracy of 77.3%, and a precision of 67.7%, indicating it is unsuitable for imbalanced datasets. In comparison, Gini Impurity was the best criterion, followed by Entropy, while Classification Error was the weakest.

Chapter 6: **Software Engineering**

6.1 Methodology

I have utilised an iterative development methodology inspired by the agile approach. This approach means I have broken down the project into smaller and manageable parts, such as experimental data analysis, data preparation, creating models, and model evaluation on my churn dataset. Each phase informed the next, creating a cohesive development pipeline. An example would be the information gained from explorative data analysis of missing data and correlated features, which strongly influenced my decisions in the data pre-processing stage.

Furthermore, I have employed an agile testing strategy by conducting continuous tests. These tests ranged from unit testing for individual classes to testing models on smaller, well, benchmarked datasets.

6.2 Git

Git was essential to the development of my project. It allowed me to manage my code, and since I was utilising RHUL GitLab, it also allowed my supervisor to access my repository for review and feedback easily. I have also followed recommended techniques for version control, such as utilising separate branches in my development, which is evident on my GitLab project page. It ensured encapsulation for developing and testing different models, creating a cleaner workflow.

I followed a conventional commit format of CommitLint[12] for my commits. This made understanding and filtering through git commits much easier and possibly helped if I needed to cherry-pick any commits. In the future, I will also consider utilising Tags and release branches, as these are considered best practices for version control and project management.

6.3 Test Driven Development

TDD is the methodology I have used to ensure that my models and classes work well. The inbuilt Python unittest module was utilised to ensure that individual components of my project work, specifically my model and evaluation metrics classes. Since I am implementing the machine learning algorithms from scratch, ensuring they are working correctly was key. Therefore, I have compared my scratch model and well-developed SKLEARN[13] existing models on a small dataset. The results from the comparison allowed me to check for any abnormal results and the need for further individual testing of any models.

6.4 Documentation

Documentation is key for many projects as it aids readers in understanding the functionality of code and why certain design decisions were taken. There are many types of documentation, such as comments, UML diagrams to visualise the system's architecture, and external guides or README files. So far, I have included detailed comments throughout most of the codebase, a README and a requirement text file to list relevant dependencies. This ensures clarity and would aid me in future development in the 2nd semester in the case of refactoring or adding to the codebase.

Appendix: Diary

Project Diary

Date: 1th October 2024

Summary

• Today, I concentrated on determining the specific sub-problem I will focus on for comparing ML algorithms. After some consideration, I decided to pursue churn prediction.

• I also began evaluating which datasets i would be using in this project. In contention between Telecom's data set and Bank dataset.

Next Steps

• Continue researching and start writing up a project plan

• Research some basic machine learning models, such as nearest neighbours, to establish a foundational understanding.

• Continue exploring potential datasets for benchmarking and implementation in the churn prediction model.

Date: 4th October 2024

Summary

• I looked into research papers on machine learning algorithms and churn prediction to gain a wider understanding and understand on how to approach my fyp.

• Initial idea i have is to have multiple different ml models to be used to predict churn such as decision tree, Logistic regression , support vector machines. Then to evaluate them empirically and give some findings on each model.

Next Steps

• Continue researching and writing up the project plan

• Discussed project plan and my churning sup problem with the supervisor before the project plan deadline.

Date: 8/10/24

Summary

•Completed the first draft of the project plan. I also created a set of question to be asked for supervisors meeting

Date: 10/10/24

Summary

•I met with my supervisor, discussed the project plan, and got feedback on my questions. I then went on to finalise the project plan and submit it.

Date: 14/10/24

Summary

• I have selected the telecom dataset for my project and have been researching exploratory data analysis (EDA) to understand its importance..

• I plan to carry out EDA on the dataset, and in the meantime, I have been learning about the Seaborn library and how it can be utilised effectively for visualising data during EDA.

Date: 19/10/24

Summary

• I have begun conducting exploratory data analysis (EDA) on the telecom dataset.

• I carried out observations on the distribution of missing data using a heatmap.

• Might run into issues later, since there are a number of rows with missing data.e.g internet usage and device protection. This would mean it would be harder to establish credible relationships between features hindering churn prediction performance potentially

Date: 21/10/24

Summary

Today, I continued Exploration Data analysis on the Telecom dataset. I utilised the Seaborn library to observe the distribution of churn categories visually and made my observations. Similarly, I made similar observations for other continuous variables, plotting histograms to understand the data better.

However, given that the dataset contains quite a lot of missing data values, I'm planning on changing datasets and researching alternative datasets that might offer more complete information. It would mean I would have to redo EDA, but it should be easier now.

Date: 22/10/24

Summary

I have selected a new dataset by the company IBM that can be downloaded from Kaggle. The dataset is a telecom dataset, and I have started EDA again. It was much easier this time as I had good practice with the previous dataset. The dataset did not have much missing data, and we dealt with it by deleting those rows. I had the choice of using the median values of the row to replace the missing data but opted to delete those rows as it was a deficient number(11 rows). I ran into some issues with the types of some of the dataset as some where object when it should have been float. Made me realise how important EDA is as it prevents errors later on when building Ml models.

Date: 26/10/24 - 30/10/24

Summary

Over these last few days, I decided to get my EDA completed, since the goal is to move on to actually implement ML algorithms that predict churn. First, I started investigating which customer attributes might have decent or strong correlation to churn. For each attribute, visual diagrams were created, such as boxplots and histograms, in order to discover some underlying relations between attributes.

I also did deep analysis with Demographics and Payment Types. From the customers, whether SeniorCitizen and Partner, based on these two statuses, I found some trends in the Churn behavior. Then, I moved my attention to features regarding service. I analyzed features like Online Security, Tech Support, Phone Service, and Multiple Lines to determine whether the kind of services provided had some impact on the churning rate. This helped me highlight customers who could be in danger of churning because of a lack of additional services. I also explored Internet Service types and Payment Methods to see if there are any service types or payment preferences that could be related to churning. An example of observation i made was that the customers using an electronic check as a payment method showed higher churn. Based on that it appears there is some relationship to the billing types and customer satisfaction.

Having completed the EDA, it gave insight into the structure of the data and relationships that existed. That allowed me to get an idea of what features may play an important role in predicting churn. That would help me with the next part of my development, building ML models and feature engineering.

Date: 10/11/24

Summary

I have been preoccupied the last few days with assignments and didn’t get to do as much work as I liked on my final year project, and now I need to catch up on my timeline. Today, I began my data preprocessing stage. I realised a mistake in my Explorative Data Analysis, where I deleted 11 rows of data because the total charge was missing. While this is partially correct, each data object has valuable information that can aid churn prediction modelling. To combat this, I utilised how long each customer stayed with the company and their monthly charge to manually compute the total charge.

After this, I decided to investigate how to convert the data for ML model as some of the data was non-numerical, such as payment methods. This is where I then learnt about encoding data and the various types of encodings. Given my dataset, I utilised binary encoding for binary variables (e.g., Yes/No or Male/Female categories) and one-hot encoding for categorical variables with multiple unique values (such as PaymentMethod or InternetService). Overall, today I have set a solid foundation for model training and will be moving on to normalisation/scaling next which is key for my first ML algo KNN.

Date: 11/11/24

Summary

Today, I completed my data preprocessing process. I did this by normalising the data and creating 2 different normalised versions of the dataset. One is the min_max normalised date, while the other is the z_score dataset. I did this to test which normalised dataset performs better with my KNN model. After this, I began developing the KNN model from scratch, utilising no libraries and using Euclidean distance as the distance metric. My initial implementation was working very slowly. This was mainly due to how I was sorting computing distances utilising insertion sort, which has a time complexity of O(n^2), which could be better given that my dataset was 7000+.

After some research, I learnt about heapq, which has a time complexity of O(logn). This reduced my computation time from 20 minutes to about 1-2 minutes, which is a huge accomplishment. I also learnt how important optimisation is with ML models.

Date: 12/11/24 - 15/11/24

Summary

I began testing out the KNN models I created on my dataset. Initially, I ran my KNN with k set to 1 neighbour and tested it on both min_max and z_score normalised churn datasets. The results were relatively okay, with an error rate of about 0.23, which is understandable since KNN is a lazy algorithm. I then ran my algorithms on larger numbers of K to find the optimal k and plotted a bar chart of error results.

here the optimal k value is 34 due to having lowest error rate.

However, I had some slight issues of rerunning models each time I open up my notebook which would be time consuming. I began investigating ways to save models such that i would not require retraining each time. This is where I found pickle, a Python library that allows saving and reloading objects such as models or evaluation metrics. This allowed me to focus on interpreting the results and visualising the error rates without redundant computations especially when testing different configurations or normalisation methods. Overall, my workflow has become more optimised compared to before.

Date: 18/11/24

Summary

After this, I, then decided to implement a known conformal predictor. This allowed me to quantify uncertainty and assign confidence levels to predictions. However my implementation could be more efficient as it takes a long time to run on a large dataset so I have decided to test on the subset of the dataset(3500). This required me to go through Vladimir Vovk's research work on conformal predictors, which was interesting.

My implementation was a bit crude and could be improved with some optimisation as the run time was very long, and this could be due to repetitive copying of conformity score. However, I did improve its optimality by only recomputing conformity score when necessary, thus reducing run time greatly from previous runs.

Date: 20/11/24

After some research on evaluation metrics for classification, I have realised that the current metrics I'm using, accuracy and error rate, are very weak. So, I have decided to use the metrics F1-score, accuracy, recall, precision, and F2-score. I created these from scratch, utilising the knowledge gained. This would help a lot since my dataset is imbalanced and give further insights, such as the number of false negatives and false positives. Utilising my new metrics class, I then updated my main Jupyter notebook to have much more in-depth results and added graphs. Overall, I am making some progress, but I should investigate/research more before coding or developing so that I wouldn't have to refactor too much.

Date 22/11/24

I have researched more about software engineer methodologies in ML and based on that I have decided to implement unit test cases for classes which is what I have done. These test classes were added for conformal and KNN classes and are fully working.

My knn is model is fully complete now and have merged into main branch with no issues.

Date 26/11/24

I have been watching videos and looking into papers about decision tree models in the recent days. I specifically focused on different uniformity measures, utilising this knowledge I then implemented a decision tree class, haven't fully tested it on my data but will do that soon. I initially started with a basic Decision tree class but realised to accommodate for different uniformity measures I would have to split into 2 classes and adjust the parameters. Looking at my original plan I have overestimated the amount of work I could do, this mainly due to workload of other modules. I am a bit behind but not so much

I have also booked a meeting with my supervisor again tomorrow and this is mainly to establish if I'm on track and to answer any questions I have.

Date 27/11/24

Today I attended my meeting with my supervisor, and I have completed a good amount of work for interim submission. I also managed to discuss what should be in the report and how I could go about presenting my FYP later on.

Furthermore, I have carried out initial testing for my Decision tree class and is preforming relatively well, some higher scores compared to knn.

Date 05/12/24 - 09/2/24

I have been really sick these past couple of days. It is a shame, considering I wanted to improve the validity of my models by adding cross-validation and perhaps adding decision boundary results for my different models. This has set me back as now I need to complete my report writing; this also meant I don't think I would be able to do my SVN model or basic GUI. However, on the bright side, I have identified and fixed a key issue; this was the fact that I was normalising the whole dataset and then splitting the churn dataset for testing and training. This had led to data leakage, which affected the validity of my results. I fixed this issue by splitting and then normalising my dataset. This has led to slightly lower results in my decision tree but hasn't affected my KNN model results.

Furthermore, in this time period I have finished my testing of decision tree model and am currently writing up my report/presentation.

Bibliography

[1] Oliver Wyman. (2004). *Customer churn*. Available at: https://www.oliverwyman.de/content/dam/oliver-wyman/global/en/files/archive/2004/CMMJ17_Customer_Churn.pdf

[2] Outsource Accelerator. "Crucial Customer Retention Statistics to Know in 2024." Outsource Accelerator, 2024. https://www.outsourceaccelerator.com/articles/customer-retention-statistics

[3] Tessitore, Sabrina. "What's the Average Churn Rate by Industry?" *CustomerGauge*, 2022. Available at: https://customergauge.com/blog/average-churn-rate-by-industry.

[4] Reichheld, F. F., & Schefter, P. (2000). *E-Loyalty: Your Secret Weapon on the Web*. Harvard Business Review.

[5] Gefen, D. Customer Loyalty in e-Commerce. J. Assoc. Inf. Syst. **2002**

[6] Morrison, J., & Vasilakos, A. (2018). An empirical comparison of techniques for the class imbalance problem in churn prediction.

[7] IBM (2017). *Telco Customer Churn*. [online] www.kaggle.com. Available at: https://www.kaggle.com/datasets/blastchar/telco-customer-churn.

[8] Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann.

[9] Max Khun and Johnson K. *Applied Predictive Modelling* (2013)

[10] Cunningham, P. and Delany, S.J. (2021). k-Nearest Neighbour Classifiers - A Tutorial. *ACM Computing Surveys*, 54(6), pp.1–25.

[11] Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

[12] CommitLint. URL: https://github.com/conventional-changelog/commitlint.

[13] Scikit-learn, URL: https://scikit-learn.org/1.5/supervised_learning.html