Comp1804 report: Predicting the topic of customers' banking questions

ENTER YOUR STUDENT ID ONLY

April 1, 2023

Word count: 3,984 words

Executive summary

This report aims to enhance online customer service in the banking sector by automating the initial filtering process of customer queries. The primary objective was to predict the subject of customer inquiries in an online chat service using machine learning.
To achieve this, we utilized traditional machine learning algorithms, including Support Vector Machines (SVM), Random Forest, Naive Bayes, AdaBoost, and Gradient Boosting, as well as neural network algorithms, such as Long Short-Term Memory (LSTM), Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Bidirectional LSTM (BiLSTM). We also performed hyperparameter tuning via grid search and cross-validation, and we evaluated the models' performance based on accuracy, precision, and recall.

Our findings revealed that the SVM model outperformed the traditional methods with an accuracy of 92.66%, while LSTM yielded the highest accuracy among the neural networks at 89.57%.
Overall, our experiments demonstrated the potential of machine learning in predicting the subject of customer inquiries in online chat services and the superiority of neural networks over traditional methods with larger datasets and computational power. These findings are significant in enhancing customer satisfaction and reducing wait times for customers seeking assistance in online chat services in the banking industry.

1 Exploratory data analysis

Exploratory data analysis (EDA) was performed to gain a deeper understanding of the data and identify any patterns, trends, or anomalies in the data. Initially, the label field of the dataset had eight distinct values due to inconsistencies in case. To resolve this issue, the labels were converted to lowercase, resulting in only four unique values. Next, the null values were visualized and it was discovered that the label field had 521 missing entries. However, there were no NaN values in the query_index and text fields. All entries in the query_index field were found to have unique values in the dataset.
The distribution of labels was then plotted using value counts, with 'card_queries_or_issues' at 19.1%, 'top_up_queries_or_issues' at 12.5%, 'needs_troubleshooting' at 31.7%, and 'other'

at 36.7%. A pie chart was also used to visualize the percentage distribution.
The average number of words per label distribution was plotted to provide additional insight. Additionally, the top 25 most frequent words in each label category were listed after some basic preprocessing such as stop word removal and lemmatization. Finally, word clouds were created for each label category to provide a more visual representation of the most frequent words.

2 Data pre-processing

The project began with pre-processing, cleaning, and splitting the dataset into training, validation, and testing sets. The initial data cleaning process involved checking the dataset's shape, column names, and value counts for the label column. Duplicate text values were checked and removed, and missing values were filled with the corresponding label group value to ensure consistency in the label column.
Next, the dataset was split into training, validation, and testing sets with a 80/20 ratio. The training set was used to train the model, the validation set was used to evaluate the model's performance during training, and the testing set was used to evaluate the final performance of the model. The text column was preprocessed to remove unwanted characters, convert text to lowercase, and remove stopwords using the NLTK library. Additionally, a new column was added to count the number of words in each text value. Lemmatization was performed on the text column to reduce words to their base form.
To prevent class imbalance, both oversampling and undersampling techniques were used, and multiple models were trained to compare their performance. Normalization, standardization, and feature encoding were not applied since they were not deemed necessary for this project. The choice to fill NaN values with the corresponding label group value was made to ensure that the missing data did not affect the overall performance of the model.
The design choices made in this project were based on the need for consistency in the label column, the need to preprocess the text column to remove unwanted characters and stopwords, and the need to balance the dataset to prevent class imbalance. The use of the NLTK library to preprocess the text column was based on its effectiveness in text preprocessing. These steps were taken to ensure that the data was properly cleaned, preprocessed, and split in a way that optimized model performance.

3 Classification using traditional machine learning

After trying different traditional machine learning models and hyperparameters, I decided to use a Support Vector Machine (SVM) classifier model as it provided the best performance on the validation set. The final hyperparameters of the model, are shown in Table 3 below:

Hyperparameter	Value
C	10
Gamma	scale
Kernel	rbf

Table 1: Hyperparameters of the Support Vector Machine classifier model

3.1 Algorithm Explanation

The Support Vector Machine (SVM) classifier model is a type of classification model that aims to find a hyperplane that separates data points of different classes with the largest margin. SVMs are particularly useful in high-dimensional spaces and are commonly used for text classification tasks.

The SVM model consists of several hyperparameters that can affect its performance, such as the regularization parameter (C), the kernel function, and the kernel coefficient (gamma).

The input text data is first preprocessed and transformed into a dense vector representation using the TF-IDF vectorizer. The SVM model is then trained using the resampled training data and the best hyperparameters obtained from the grid search. During training, the SVM model aims to find the optimal hyperplane that maximizes the margin between the two classes.

3.2 Experiments to optimize the model

Hyperparameter Optimization

To optimize the hyperparameters of our SVM model, we employed a combination of manual tuning and GridSearchCV. Firstly, we manually tried different combinations of hyperparameters and evaluated the performance of the model on the validation set. Based on these results, we selected a set of hyperparameters that provided the best performance and used them as the starting point for the GridSearchCV.

We chose to use an SVM model as it is a widely used and effective method for classification tasks. SVMs are particularly useful for text classification tasks because they can handle high dimensional data well and are able to find non-linear decision boundaries.

We used the TF-IDF vectorizer to convert our text data into numerical feature vectors. TF-IDF stands for term frequency-inverse document frequency, and it is a commonly used method for feature extraction in text analysis. It assigns a weight to each word based on how often it appears in the document and how often it appears across all documents in the corpus.

We used a C value of 10 and gamma value of 'scale' for our SVM model. The C parameter controls the trade-off between misclassification of training examples and simplicity of the decision surface. We chose a value of 10 as it provided the best performance on our validation set.

The gamma parameter defines how far the influence of a single training example reaches. A low value of gamma will consider only the nearby points in calculating the decision boundary, while a high value of gamma will consider all the points in the training set. We chose a value of 'scale' for gamma as it is the default value and worked well for our dataset.

We used the radial basis function (RBF) kernel for our SVM model. The RBF kernel is a popular kernel function that is commonly used in SVM. It is a non-linear kernel that maps the input space to a high-dimensional feature space, allowing for the modeling of non-linear decision boundaries.

Overall, our design choices were based on a combination of theoretical understanding and experimentation with various hyperparameters. We selected values that have been shown to work well in similar text classification tasks, and we performed a rigorous grid search to optimize these hyperparameters. This allowed us to find the combination of hyperparameters that gave us the best accuracy on our validation set.

3.3 Comparison with other models

3.3.1 Naive Bayes Model

Naive Bayes is another traditional machine learning model used for text classification. It is a probabilistic model that assumes independence among the features. It was implemented using the scikit-learn library and a TfidfVectorizer was used for feature extraction. The best hyperparameters were selected using a grid search with cross-validation. The model achieved an accuracy of 82.88% on the test set.

3.3.2 AdaBoost Model

AdaBoost is an ensemble learning algorithm used for binary classification. It works by combining multiple weak classifiers to form a strong classifier. It was implemented using the scikit-learn library and a TfidfVectorizer was used for feature extraction. The best hyperparameters were selected using a grid search with cross-validation. The model achieved an accuracy of 58.54% on the test set.

3.3.3 Random Forest Model

Random Forest is another ensemble learning algorithm used for text classification. It works by constructing multiple decision trees and combining their outputs to make the final prediction. It was implemented using the scikit-learn library and a TfidfVectorizer was used for feature extraction. The best hyperparameters were selected using a grid search with cross-validation. The model achieved an accuracy of 81.89% on the test set.

3.3.4 Gradient Boosting Model

Gradient Boosting is another ensemble learning algorithm used for text classification. It works by sequentially adding new models that correct the errors made by the previous models. It was implemented using the scikit-learn library and a TfidfVectorizer was used for feature extraction. The best hyperparameters were selected using a grid search with cross-validation. The model achieved an accuracy of 86.70% on the test set.

3.4 Model Evaluation

We used five models to classify our data: a naive bayes model, a support vector model, an adaboost model, a random forest model and a gradient boosting model. To evaluate the performance of these models, we used the following methods:

We evaluated the performance of each model using a confusion matrix and performance metrics: accuracy, precision, and recall. The confusion matrix showed that all models were able to correctly classify the majority of the samples, with varying levels of false positives and false negatives. The accuracy metric measures the overall percentage of correctly classified samples, while the precision measures the proportion of correctly classified positive samples and recall measures the proportion of positive samples that were correctly classified. The metrics were high for the SVM model, indicating that it performed well on the task.

Finally, we compared the performance of each model to a "trivial" baseline that always predicted the majority class. All models significantly outperformed this baseline, demonstrating that they were able to learn meaningful patterns in the data and make accurate predictions.

3.4.1 Comparison of Model Performance on Undersampled Dataset

Table 2 shows the performance metrics of various traditional machine learning models on an undersampled dataset. The SVM model achieved the highest accuracy of 0.9003, followed by the Gradient Boosting model with an accuracy of 0.8582. The Adaboost model achieved the lowest accuracy of 0.5953. The precision and recall values for all models were relatively similar, with the SVM model having the highest precision of 0.9022 and the highest recall of 0.9102.

Model	Accuracy	Precision	Recall
Naive Bayes	0.8200	0.8145	0.8427
SVM	0.9003	0.9022	0.9102
Adaboost	0.5953	0.6077	0.5991
Random Forest	0.8086	0.8140	0.8299
Gradient Boosting	0.8582	0.8642	0.8796

Table 2: Performance of traditional machine learning models on an undersampled dataset

3.4.2 Comparison of Model Performance on Oversampled Dataset

Table 3 shows the performance metrics of various traditional machine learning models on an oversampled dataset. The SVM model achieved the highest accuracy of 0.9266, followed by the Gradient Boosting model with an accuracy of 0.8670. The Adaboost model achieved the lowest accuracy of 0.5854. The precision and recall values for all models were relatively similar, with the SVM model having the highest precision of 0.9294 and the highest recall of 0.9288.

Model	Accuracy	Precision	Recall
Naive Bayes	0.8288	0.8268	0.8433
SVM	0.9266	0.9294	0.9288
Adaboost	0.5854	0.6220	0.5784
Random Forest	0.8189	0.8270	0.8372
Gradient Boosting	0.8670	0.8728	0.8753

Table 3: Performance of traditional machine learning models on an oversampled dataset

From the results, we can observe that the SVM model performed consistently better than other models on both oversampled and undersampled datasets. The Adaboost model performed poorly on both datasets. We observed that oversampling the dataset resulted in improved performance for all models, with the SVM and gradient boosting models achieving the highest accuracies.

4 Classification using neural networks

After trying different neural network architectures and hyperparameters, I decided to use a Bidirectional LSTM model as it provided the best performance on the validation set. The final hyperparameters of the model, are shown in Table 4 below:

Hyperparameter	Value
Vocabulary size	50,000
Embedding size	128
LSTM size	64
Dropout rate	0.2
Dense size	128
Number of epochs	10
Batch size	32

Table 4: Hyperparameters of the Bidirectional LSTM model

4.1 Algorithm Explanation

The Bidirectional LSTM model is a type of Recurrent Neural Network (RNN) that has been shown to perform well on sequential data such as text. The key idea behind the Bidirectional LSTM model is to process the input sequence in both forward and backward directions using two separate LSTM layers. This allows the model to capture both the past and future context of each word in the input sequence, which can help it better understand the overall meaning of the text.

The model consists of several layers, including an Embedding layer, two LSTM layers, a Dropout layer, and a Dense layer. The Embedding layer is used to transform the input text data into a dense vector representation that can be understood by the LSTM layers. The LSTM layers process the input sequence in both directions, with the output from each direction concatenated together at each time step. The Dropout layer is used to prevent overfitting by randomly dropping out some of the nodes in the network during training. Finally, the Dense layer maps the output from the LSTM layers to the output classes using a softmax activation function.

4.2 Experiments to optimize the model

Hyperparameter Optimization

To optimize the hyperparameters of our Bidirectional LSTM model, we employed a combination of manual tuning and GridSearchCV. Firstly, we manually tried different combinations of hyperparameters and evaluated the performance of the model on the validation set. Based on these results, we selected a set of hyperparameters that provided the best performance and used them as the starting point for the GridSearchCV.

We chose to use a Bidirectional LSTM model as it has been proven to outperform traditional LSTMs in various natural language processing tasks. By processing the input sequence in both directions, the Bidirectional LSTM is capable of accessing past and future information simultaneously, resulting in a better understanding of the input sequence.

We also chose to use a high number of words (50,000) to represent the text, as our dataset was large and contained a wide range of vocabulary. We found that using more words in the vocabulary improved the accuracy of the model.

We chose a LSTM size of 128 because it has been shown to work well in similar natural language processing tasks. We also chose an embedding size of 128 because it was the same as the LSTM size.

To prevent overfitting, we used a dropout rate of 0.2. Dropout randomly drops out a certain percentage of units in the neural network during training, forcing the remaining units to learn more robust features that are more generalizable to new data.

We chose a dense size of 128 as we found that a larger dense layer improved the accuracy of the model.

Finally, we used 10 epochs and a batch size of 32, as these values gave us the best balance between training time and model accuracy.

To optimize our hyperparameters further, we used GridSearchCV to perform a cross-validated grid search over a range of values for each hyperparameter. This allowed us to find the combination of hyperparameters that gave us the best accuracy on our validation set.

Overall, our design choices were based on a combination of theoretical understanding and experimentation with various hyperparameters. We selected values that have been shown to work well in similar natural language processing tasks, and we performed a rigorous grid search to optimize these hyperparameters for our specific dataset.

4.3 Comparison with other models

4.3.1 RNN Model

The RNN model was created using Keras library and consisted of an Embedding layer, a SimpleRNN layer, a Dropout layer, and a Dense layer. The input to the model was the sequence of word indices with a maximum sequence length of 100. This model was trained on the training set and evaluated on the validation set. The results showed that the RNN model achieved an accuracy of 87.47% on the validation set.

4.3.2 CNN Model

The CNN model was created using Keras library and consisted of an Embedding layer, two Conv1D layers, two MaxPooling1D layers, a GlobalMaxPooling1D layer, a Dropout layer, and two Dense layers. The input to the model was the sequence of word indices with a maximum sequence length of 100. This model was trained on the training set and evaluated on the validation set. The results showed that the CNN model achieved an accuracy of 88.84% on the validation set.

4.3.3 LSTM Model

The LSTM model was created using Keras library and consisted of an Embedding layer, an LSTM layer, a Dropout layer, and two Dense layers. The input to the model was the sequence of word indices with a maximum sequence length of 100. This model was trained on the training set and evaluated on the validation set. The results showed that the LSTM model achieved an accuracy of 89.45% on the validation set.

4.4 Model Evaluation

We used four models to classify our data: a simple RNN model, a CNN model, an LSTM model, and a bidirectional LSTM model. To evaluate the performance of these models, we used the following methods:

We evaluated the performance of the BiLSTM model using a confusion matrix and two performance metrics: accuracy and F1 score. The confusion matrix showed that the model was able to correctly classify the majority of the samples, with only a small number of false positives and false negatives. The accuracy metric measures the overall percentage of correctly classified samples, while the F1 score measures the balance between precision (the proportion of correctly classified positive samples) and recall (the proportion of positive samples that were correctly classified). Both metrics were high for the BiLSTM model, indicating that it performed well on the task.

Finally, we compared the performance of the BiLSTM model to a "trivial" baseline that always predicted the majority class. The BiLSTM model significantly outperformed this baseline, demonstrating that it was able to learn meaningful patterns in the data and make accurate predictions.

4.4.1 Comparison of Model Performance on Undersampled Dataset

Table 5 shows the performance metrics of the four models on an undersampled dataset. The CNN model achieved the highest accuracy of 0.8563, while the RNN model achieved the lowest accuracy of 0.8334. The precision and recall values for all models were relatively similar, with the CNN model having the highest precision of 0.8730 and the highest recall of 0.8579.

Model	Accuracy	Precision	Recall
RNN	0.8334	0.8350	0.8376
CNN	0.8563	0.8730	0.8579
LSTM	0.8502	0.8535	0.8563
BiLSTM	0.8494	0.8589	0.8533

Table 5: Performance of models on an undersampled dataset

4.4.2 Comparison of Model Performance on Oversampled Dataset

Table 6 shows the performance metrics of the four models on an oversampled dataset. The BiLSTM model achieved the highest accuracy of 0.8957, followed closely by the LSTM model with an accuracy of 0.8945. The RNN model achieved the lowest accuracy of 0.8747. The precision and recall values for all models were relatively similar, with the BiLSTM model having the highest precision of 0.9010 and the highest recall of 0.9029.

Model	Accuracy	Precision	Recall
RNN	0.8747	0.8829	0.8841
CNN	0.8884	0.8979	0.8948
LSTM	0.8945	0.8958	0.9005
BiLSTM	0.8957	0.9010	0.9029

Table 6: Performance of models on an oversampled dataset

We observed that oversampling the dataset resulted in improved performance for all models, with the BiLSTM and LSTM models achieving the highest accuracies. The CNN model also showed a significant improvement in accuracy with the oversampled dataset. In contrast, undersampling the dataset resulted in lower performance for all models, with the CNN model achieving the highest accuracy in this case. These results highlight the importance of carefully selecting and preprocessing the dataset for training machine learning models.

5 Ethical discussion

There are several social and ethical implications of implementing a machine learning system for predicting the topic of customer banking questions in an online chat service. We can discuss these implications using the Ethical OS Toolkit.
Privacy: Data collection and processing raise concerns about the privacy of customer information. There is a risk that sensitive personal information could be leaked or used for other purposes without the customer's consent. Therefore, it is important to ensure that the data collected is used only for the intended purpose, and that appropriate security measures are in place to protect customer privacy.
Bias: Machine learning systems are only as unbiased as the data they are trained on. Therefore, there is a risk of introducing bias if the training data is not diverse or representative of the population. This could lead to unfair treatment of certain groups of customers, and undermine the credibility of the system. To address this, it is important to ensure that the training data is diverse and representative of the population.
Accountability: It is important to ensure that the decisions made by the machine learning system are transparent and accountable. Customers should be informed about the use of the system and the criteria used for predicting the topic of their questions. In addition, there should be a clear process for addressing any issues or complaints that arise as a result of the system's predictions.
Accessibility: It is important to ensure that the machine learning system is accessible to all customers, regardless of their level of digital literacy or ability. This could be achieved by providing clear instructions and support for customers who may have difficulty using the system.
Impact on employees: The implementation of a machine learning system could have an impact on the roles and responsibilities of employees who previously handled customer questions manually. It is important to ensure that employees are informed about the system and given appropriate training to work with it effectively.

6 Recommendations

- The best candidate for the task of predicting the topic of customer banking questions is the Support Vector Machine (SVM) classifier. The SVM model outperformed other traditional machine learning models as well as neural networks in terms of accuracy, precision, and recall. It is also known for being robust against overfitting, which is important for generalizing the model to new data.
- The final SVM model achieved an accuracy of 92.66%, which is a good performance for a text classification task. However, it is important to note that this accuracy was achieved on a specific dataset, and performance on new data may vary. Therefore, further testing and validation on new data is necessary before deploying the model in practice.
- Our top suggestion for future improvements is to explore the use of transfer learning to improve the performance of the neural network models. Transfer learning involves pretraining a neural network on a large dataset, and then fine-tuning it on a smaller dataset for a specific task. This could help address the issue of overfitting and improve the accuracy of the neural network models. Another suggestion is to collect and label more data, which can help improve the diversity and representativeness of the training dataset, and thereby improve the generalization performance of the model.

7 Retrospective

Reflecting on my coursework, if I were to start this coursework again, I would delve deeper into investigating advanced methods for tuning hyperparameters, such as Bayesian optimization or genetic algorithms. Additionally, I would explore expanding evaluation metrics beyond accuracy to gain a more comprehensive understanding of model performance.

References

D. Cer, Y. Yang, S. yi Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil. Universal sentence encoder, 2018.

S. Chowdhury and M. P. Schoen. Research paper classification using supervised machine learning techniques. In *2020 Intermountain Engineering, Technology and Computing (IETC)*, pages 1–6, 2020. doi: 10.1109/IETC47856.2020.9249211.

J. Khairnar and M. Kinikar. Machine learning algorithms for opinion mining and sentiment classification. 2013.

Y. Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, Oct. 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1181. URL https://aclanthology.org/D14-1181.

K. Mysiak. Nlp part 3 | exploratory data analysis of text data. https://towardsdatascience.com/nlp-part-3-exploratory-data-analysis-of-text-data-1caa8ab3f79d, 2019.

S. Patel. Chapter 2: Svm (support vector machine)—theory. https://medium.com/machine-learning101/chapter-2-svm-support-vector-machine-theory-f0812effc72, 2017.