VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master's thesis

Sentiment Analysis: Evaluating model performance and transferability across different sectors

Sentimentų analizė: modelių našumo ir pernaudojimo skirtingose platformose vertinimas

Emilija Rizgelytė

Supervisor : Asist. Dr. Andrius Buteikis

Vilnius
2026

Acknowledgements

The author is thankful to the Information Technology Research Center, Faculty of Mathematics and Informatics, Vilnius University, for providing the High-Performance Computing (HPC) resources for this research and to supervisor Andrius Buteikis for his guidance on research direction and scope, support, advices and constructive feedback.

Summary

Online reviews and social media posts strongly influence consumer decision making, motivating the need for reliable sentiment analysis models. Rising variety of different platforms storing sentiment data introduces the need of model transferability, to have reliable sentiment analysis models when applied to text originating from different platforms. This thesis investigates cross-domain generalisability of sentiment analysis by comparing model performance when training and testing data come from distinct sources. Five datasets are used: IMDB and Rotten Tomatoes movie reviews, Amazon product reviews, Spotify app reviews from Google Play store and Twitter posts (tweets). Text in each dataset is preprocessed (lowercasing, HTML and non-alphabetic removal, whitespace normalisation, tokenisation, and stop word removal) and represented using Bag of Words, TF-IDF, and Word2Vec embeddings. Classical supervised models (Naïve Bayes, Logistic Regression, linear SVM) and an LSTM based recurrent neural network are evaluated, alongside support vector regression. The experiments include testing models within the same dataset, testing how well a model trained on one platform works while tested on another, removing platform or domain-specific words to see how this affects model performance, training on several datasets combined (expanding training dataset to have context from various platforms), and improving predictions by averaging results from multiple models built with the same feature representation. Results show consistent performance degradation under cross-platform transfer, with the magnitude depending on platform and text style: models trained on Amazon and Spotify transfer comparatively better, while models trained on Twitter generalise worst, and transfer between the two movie review platforms is weaker than domain similarity would suggest. TF-IDF and Bag of Words provide strong and stable sentiment analysis model baselines, Word2Vec performs well only for selected dataset pairs, and the LSTM model is competitive but does not eliminate domain shift. Domain-specific word removal and naive dataset aggregation yield mixed, generally small effects, and ensembles provide only marginal and inconsistent improvements. Friedman and Nemenyi statistical tests indicate that while many cross-platform models differ significantly from intra-domain ones, no robust, consistent benefits arise from domain word removal or ensemble averaging. Overall, the findings indicate that platform mismatch is a primary limitation for sentiment model portability, and that effective cross-platform adaptation likely requires more advanced domain adaptation or representation learning techniques beyond the baseline strategies explored here.

Keywords: Sentiment analysis, natural language processing, cross-platform transfer, ensemble averaging

Santrauka

Internetinės apžvalgos ir įrašai socialiniuose tinkluose daro didelę įtaką vartotojų sprendimų priėmimui, taip pažymint sentimentų analizės modelių poreikį. Didėjant skaičiui platformų, kuriose saugomi sentimentų duomenys, iškyla poreikis modelius perkelti, kad juos būtų galima naudoti ant skirtingų duomenų šaltinių. Šiame darbe nagrinėjamas sentimentų analizės apibendrinamumas tarp domenų, lyginant modelio našumą, kai mokymo ir testavimo duomenys gaunami iš skirtingų šaltinių. Naudojami penki duomenų rinkiniai: IMDB ir „Rotten Tomatoes“ filmų apžvalgos, „Amazon“ produktų apžvalgos, „Spotify“ programėlės apžvalgos iš „Google Play“ parduotuvės ir „Twitter“ įrašai. Kiekvieno duomenų rinkinio tekstas yra iš anksto apdorojamas (perrašoma į mažąsias raides, pašalinimi HTML ir kiti simboliai, normalizuojami tarpai, vykdomas teksto skaidymas į teksto vienetus ir atmetinių žodžių (angl. *stop words*) pašalinimas) ir pateikiamas naudojant žodžių maišo (angl. *Bag of Words*), TF-IDF ir „Word2Vec“ įterpimus. Vertinami klasikiniai modeliai (naivusis Bajeso klasifikatorius, logistinė regresija, tiesinis atraminių vektorių klasifikatorius) ir LSTM pagrįstas neuroninis tinklas, kartu su atraminių vektorių regresija. Eksperimentai apima modelių testavimą tame pačiame duomenų rinkinyje, bandymą, kaip gerai veikia vienoje platformoje apmokytas modelis, kai jis testuojamas ant kitos platformos duomenų, žodžių, susijusių su konkrečiomis sritimis, pašalinimą, siekiant pamatyti, kaip tai veikia modelio našumą, mokymą keliuose duomenų rinkiniuose kartu (išplečiant mokymo duomenų rinkinį, kad prasiplėstų kontekstas iš įvairių platformų) ir prognozių gerinimą, vidurkinant kelių modelių, sukurtų naudojant tas pačias žodžių įterpimo technikas, rezultatus. Rezultatai rodo nuoseklų našumo pablogėjimą perduodant duomenis tarp skirtingų platformų, o pablogėjimo apimtis priklauso nuo platformos ir teksto stiliaus: „Amazon“ ir „Spotify“ apmokyti modeliai perduoda duomenis palyginti geriau, o „Twitter“ apmokyti modeliai apibendrina blogiausiai, perdavimas tarp dviejų filmų apžvalgų platformų yra silpnesnis, nei galima būtų manyti pagal srities panašumą. TF-IDF ir „Bag of Words“ pateikia stiprias ir stabilias sentimentų analizės modelių bazes, „Word2Vec“ gerai veikia tik pasirinktoms duomenų rinkinių poroms, o LSTM modelis yra konkurencingas, bet nepašalina sričių skirtumų. Sričiai būdingų žodžių pašalinimas ir naivūs duomenų rinkinių agregavimai duoda mišrius, dažniausiai nedidelius efektus, o ansambliai suteikia tik nežymius ir nenuoseklius patobulinimus. Friedmano ir Nemenyi testai rodo, kad, nors daugelis skirtingų platformų modelių reikšmingai skiriasi nuo tarpsričių modelių, jokios patikimos, nuoseklios naudos iš srities žodžių pašalinimo ar ansamblinio vidurkinimo nėra. Apskritai išvados rodo, kad platformų neatitikimas yra pagrindinis sentimentų analizės modelių perkeliamumo apribojimas ir kad veiksmingai platforminei adaptacijai greičiausiai reikia pažangesnių srities adaptacijos arba reprezentacijos mokymosi technikų, be čia nagrinėjamų bazinių strategijų.

Raktiniai žodžiai: Sentimentų analizė, natūralios kalbos apdorojimas, žodžių maišas, atraminių vektorių klasifikatorius

List of Figures

List of Tables

Contents

Introduction

In recent years, the Internet has become a place where people search for information about their desired product or service, while sharing their opinion about them too. This has resulted in an increase of online comments and reviews which can have a big influence on people's demand for certain products or services. Analysing the sentiments of netizens helps businesses understand the needs and wants of the customer better, and sentiment analysis is a great way to achieve that [12]. It is a specific subtask within the broad area of opinion mining in Natural Language Processing (NLP) that classifies the text according to the emotion it appears to convey [43]. Typically, it automatically assigns the texts to positive, negative and neutral classifications and is an important part in the development of artificial intelligence [5].

Sentiment analysis (SA) has become very popular among research communities in the past decade, with various methods implemented. The most popular ones include lexicon-based approach (the oldest SA method), machine learning, deep learning classifiers, hybrid and other approaches [25]. Nowadays, most of the researchers additionally try to incorporate transfer learning approaches that would be capable of generalizing across different types of text data; however, this approach is mostly used at intra-domain level [37]. How sentiment analysis models work at cross-platform level instead of intra-domain is the main focus of this thesis as there is little to none research done on the matter.

Different websites that gather customer review data, each have specific communities that might use different text styles or slang words. Sentiment analysis models built on specific platforms may not perform well on the data from another one. On the other hand, having a generalized model that could be used on various types of text data could prove to be beneficial. This thesis aims to analyse the text properties from different sites (Amazon, IMDB, Google Play store, etc.), build various types of sentiment analysis models on the data, test the models on the data that they were built on and then train them not only on the source data, but also on the data from other sites and compare the results. The structure of the thesis is as follows:

- Section 1 outlines the research done in similar areas of sentiment analysis
- Section 2 focuses on the theoretical properties of the technologies used in the analytical part of the thesis
- Section 3 part consists of the practical tasks and their results
- Conclusions are presented in section 4 and summarise the findings of the thesis, state the problems that came along the way and describes the path for future studies

1 Literature review

As sentiment analysis became more and more popular, the number of research studies on it has increased quite a lot. Most of the papers focus on the implementation of sentiment analysis for various fields and objectives, but there are some articles that give a great overview on this area of NLP. Article [25] summarises the whole concept of sentiment analysis, describes different approaches with their advantages and drawbacks, but it does not conduct any practical analysis. Nevertheless, the article is a good material for beginners in the field of sentiment analysis or in the whole NLP area.

Text-preprocessing is one of the most important parts of data mining for sentiment analysis, but as the years go by and the technologies, along with the models that we are now capable of building, advance, some scientists have raised questions regarding the need for text preprocessing with the new tools. The answer to this question can be found in articles [39] and [24]. In the first paper [39], released in 2016, the authors try to understand and classify sentiments in Twitter tweets, focusing on text normalization to address informal language, spelling errors, and slang words. The results show the importance of preprocessing text data, as the algorithms better capture sentiments by analysing the context and significance of words, especially slang. The second paper [24], released in 2024 further indicates that text-preprocessing for sentiment analysis is still as important as it was before. The authors investigate and compare, how preprocessing impacts on the text classification performance of modern and traditional classification models. The results strongly support that choosing the best preprocessing technique, in place of the worst, can significantly improve accuracy on the classification (up to 25%). Even a simple Naïve Bayes classifier proved to outperform (i.e., by 2% in accuracy) the best performing Transformer when text preprocessing was in place.

After text preprocessing, another crucial part of the sentiment analysis is feature extraction. A study [3] compares TF-IDF and N-Grams feature extraction methods on the SS-Tweet dataset. For classification, various algorithms, including SVM, Decision Trees, and Logistic Regression, were applied and TF-IDF was found to be more effective on these algorithms by 3-4%. Both features alongside Logistic Regression emerged as the most effective for sentiment predictions across four metrics: accuracy, precision, recall, and F1 score. Even though the study lacks comparisons between other feature extraction methods, the findings indicate the importance of a good feature selection for sentiment analysis model.

To check the model adequacy, most of the researches use classification evaluation metrics for sentiment analysis - accuracy, precision, recall and F1 score. Article [33] provides an overview of common evaluation metrics and statistical tests used in various machine learning tasks, including binary, multi-class, and multi-label classification, regression, image segmentation, object detection, and information retrieval. They explain key metrics such as accuracy, sensitivity (recall), specificity, precision, F1 score, Cohen's kappa, Matthews' correlation coefficient. The paper also details statistical testing methods such as McNemar's test, DeLong test, Wilcoxon signed-rank test, and Friedman's test to compare model performances. By providing practical examples, the authors demonstrate the application of these metrics and tests, emphasizing the importance of appropriate methodological choices to avoid flawed conclusions. Another article on evaluation metrics [11], critiques commonly

used evaluation measures (Recall, Precision, F-Measure, and Accuracy) for their biases and limitations. The author argues that these measures often provide skewed assessments of a system's performance because they don't account for performance in correctly handling negative examples and chance level performance. To address these limitations, the author introduces the concepts of Informedness, Markedness, and Correlation, which offer unbiased evaluation metrics by considering both the positive and negative predictive power of a model. However, the new metrics suggested so far have only beed tested in a small area of sentiment analysis and thus will not be used in this thesis.

A lot of studies were conducted on various data sources from different business fields, e.g. products reviews from Amazon, tweets from Twitter, movie, restaurant, app reviews that mostly try to find the best models for classifying the sentiments. An article [1] used sentiment analysis to predict the polarity of Amazon mobile phone dataset reviews using supervised machine learning algorithms like logistic regression, Naïve Bayes, Random Forest, Recurrent Neural Networks, Transformers. BERT model has achieved the best result in multi-class classification and binary classification, with accuracy of 94% and 98%, respectively. Study on people sentiments on Twitter during Russian aggression in Ukraine [23] investigates the use of machine learning classifiers like K-Nearest Neighbor, Random Forest, Decision Tree, Logistic Regression, Naïve Bayes, XGBoost, AdaBoost, and Multi-layer Perceptron. Among the classifiers, the Multi-layer Perceptron model demonstrated the highest accuracy, outperforming other existing sentiment classification techniques. The study [27] focuses on the use of text mining (TM) and sentiment analysis to assess medical records of seafarers and understand their health issues. Sentiment analysis is performed using both lexicon and machine learning-based methods, with Naïve Bayes ultimately chosen for sentiment classification. Publication [15] compares Convolutional Neural Networks and Recurrent Neural Networks across various natural language processing tasks. Findings reveal that RNNs, particularly GRUs and LSTMs, perform better when task comprehension relies on understanding entire sequences or long-range semantics, whereas CNNs are effective for tasks focusing on local features or key phrases. Apart from the mentioned publications, there are many more trying to answer the same question - which sentiment analysis approach is the best for their target domain. Even though these researches bring a lot of value to businesses and the NLP area in general, the main gap of all of these articles is that they focus solely on the specific domain and do not try to generalise the findings on a bigger scale.

When it comes to aforementioned domain specific tasks, models built directly on the domain knowledge usually exceed the performance of the larger general ones. Survey [14] on LLM (Large Language Model) highlights the increasing opinion that model size alone does not guarantee domain competence - smaller, carefully designed models might outperform much larger general models when it comes to specialized tasks. Another study [31] investigates the performance of specialized small models versus general large models in text classification tasks. Key findings indicate that specialized models often need only a few samples (around 100 on average) to match or exceed the performance of general models. However, the required number of labels is highly dependent on the dataset's characteristics. Publication [41] addresses whether there is a necessity for domain-specific embedding models, particularly in the context of the finance sector. The paper concludes that despite being trained on comprehensive data sources, general LLM-based embedding models may not

fully address the specialized needs of particular domains. This underscores the importance of developing domain-specific embedding models and benchmarks to enhance NLP applications' accuracy and relevance in specialized fields. In this master's thesis, the question whether general models are outperformed by domain-specific ones in particular scenarios will be answered as well.

The most popular way to transfer sentiment analysis models currently used is transfer learning. Research [42] explores advanced sentiment analysis techniques, addressing limitations in existing Aspect-Based Sentiment Analysis methods. It proposes enhancements through transfer learning, specifically with models like BERT and XLNet, to improve sentiment classification. The approach is tested on datasets from Yelp (restaurant reviews), Wine Enthusiast, and Rotten Tomatoes (movie reviews) to cover different domains. The results demonstrate that the proposed models (especially XLNet and BERT) outperform baseline deep learning and machine learning models in terms of subset accuracy, Hamming loss, and F1 scores. Another paper [37] focuses on improving sentiment detection, particularly by addressing challenges in transferring learning techniques between different domains. It introduces a new technique called Sentiment Detection with Auxiliary Data (SDAD), which uses Kernel Density Estimation (KDE) to model joint distribution differences and better incorporate source domain examples into the sentiment detection process. The paper compares this method to traditional approaches like support vector machines (SVM) and structural correspondence learning (SCL), using product reviews. The experiments demonstrate the effectiveness of the SDAD method over other state-of-the-art methods, showing better adaptation to domain differences and improved sentiment classification results. These two articles try to generalise the sentiment analysis models between different domains, however, they do not look into the differences between different kinds of sites and internet websites that gather text data.

2 Methodology

2.1 Text pre-processing and feature extraction

In NLP data pre-processing techniques are used for removing unnecessary words, characters, or punctuation that do not contribute to machine interpretation. These techniques primarily focus on transforming raw data into a structured format that emphasizes the keywords that highlight the context (sentiment) of the sentence or paragraph. Text pre-processing techniques used in this thesis:

1. Converting text to lowercase
2. Removal of HTML tags and non-alphabetic characters
3. Whitespace Normalization
4. Tokenization
5. Stop Words Removal

The first three techniques are understandable by themselves. Tokenization is the process of breaking down text into smaller units, called tokens, which can be individual words, phrases, or symbols. The splitting criteria is mainly at the occurrence of a space or a punctuation. This step helps in filtering out unwanted words in further processing steps. In Python, pre-trained models for tokenization are available in NLTK resource `punkt` [28]. Regarding stop words removal, stop words are words that are used very frequently in the language and usually have no particular meaning. They include functional words - words that express certain grammatical functions rather than specific meanings (e.g.: a, an, the, in, at, but, so...) [40]. In Python, stop words are available in NLTK resource `stopwords` [16].

Feature extraction transforms features into vector forms, making them comprehensible to machines. The techniques used for feature extraction result in each feature being represented as a vector, which can then be input into classifier models. Feature extraction methods used in this thesis:

1. Bag of Words (BOW)
2. TF-IDF
3. Word2Vec

BOW groups the features together based on the number of occurrences of the word in the document and it is not concerned where it occurs in the text. The intuition is that texts that contain the same words are similar in context. Drawbacks of the BOW model is that it gives preference to words that occur more frequently thus making them more important when they might not have information that helps to classify better. BOW logic is illustrated in Figure 1:

Document D1	*The child makes the dog happy* the: 2, dog: 1, makes: 1, child: 1, happy: 1
Document D2	*The dog makes the child happy* the: 2, child: 1, makes: 1, dog: 1, happy: 1

↓

	child	dog	happy	makes	the	BoW Vector representations
D1	1	1	1	1	2	[1,1,1,1,2]
D2	1	1	1	1	2	[1,1,1,1,2]

Figure 1. *BOW representation [4]*

TF-IDF (Term Frequency-Inverse Document Frequency) is a feature extraction model similar to BOW, but it is additionally adjusted for the fact that some words generally appear more frequently in the documents [2]. As the name suggests, the model combines two components: Term Frequency (TF) and Inverse Document Frequency (IDF). TF measures how often a word appears in a document [18]:

$$IDF(t, D) = \log\left(\frac{\text{Total number of documents in the corpus } N}{\text{Number of documents containing term } t}\right). \tag{1}$$

IDF reduces the weight of common words across multiple documents while increasing the weight of rare words:

$$TF(t, d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in document } d}. \tag{2}$$

The final TF-IDF score is the product of TF and IDF:

$$\text{TF-IDF}(t, d, D) = TF(t, d) \times IDF(t, D). \tag{3}$$

Word2Vec is a word embedding technique that allows words to be represented as vectors in a continuous vector space [10]. These models provide dense, lower-dimensional embeddings and focus more on the semantic meanings, context and usage of the words, unlike BOW and TF-IDF which focus more on word frequency and does not have the ability to understand word context. Word2Vec can use two architectures: CBOW (Continuous Bag of Words) or Skip gram. CBOW predicts the current word given context words within a specific window while Skip Gram predicts the surrounding context words within specific window given current word:

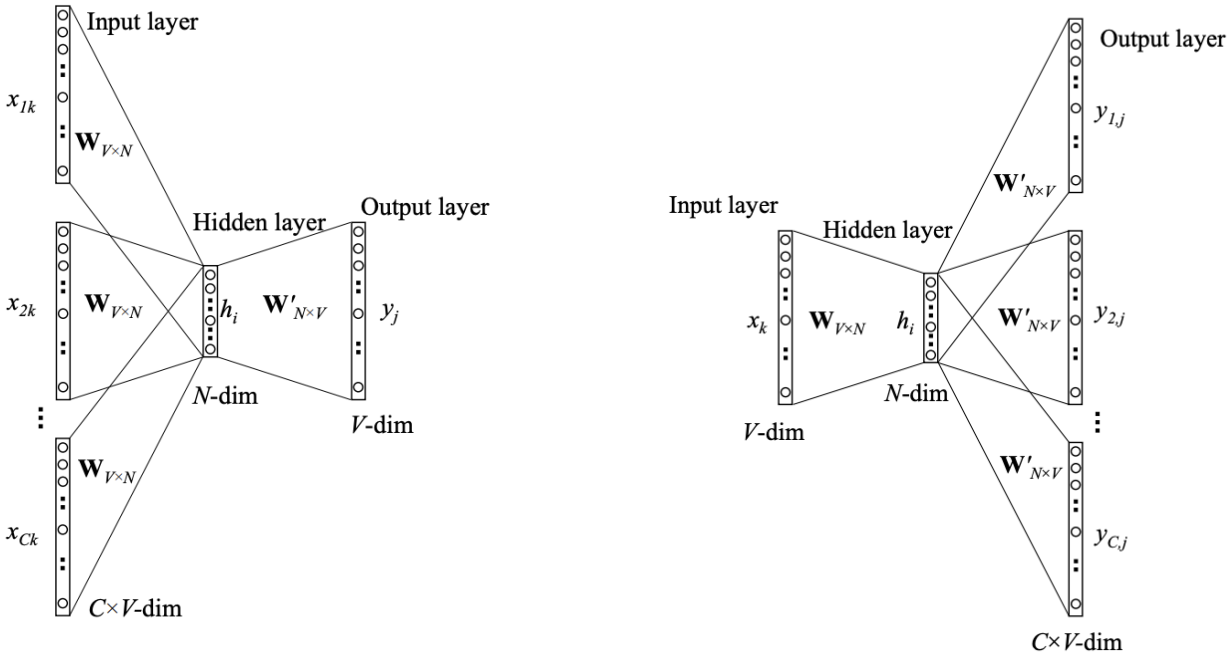

Figure 2. *CBOW (left) and Skip Gram (right) representations [35]*

2.2 Models

As mentioned in section 1 of the thesis, the field of sentiment analysis and NLP in general is evolving every day and there are many great new approaches for the topic. In this thesis, five models were used for evaluation of sentiment analysis:

- Naïve Bayes (NB) - one of the most popular choices for sentiment analysis due to easy implementation and interpretation
- Support Vector Machine (SVM) - effective model that provides good overall accuracy
- Logistic Regression (LR) - also easy to understand and implement, very popular for binary classification tasks
- Recurrent Neural Networks (RNN) - designed to capture the context and meanings of the words
- Support Vector Regression (SVR) - similar to SVM, but used for regression rather than classification, taken as an alternative approach aimed at predicting continuous values of sentiment analysis rather than binary ones

Naïve Bayes is a supervised machine learning algorithm that predicts the category of a data point using methods based on conditional probability [20]. This classifier generally performs worse than more advanced models, but it is highly scalable and interpretable, thus a popular choice for sentiment analysis. The probabilistic feature of the classifier is based on Bayes' Theorem [13]:

$$P(A \mid B) = \frac{P(A) \cdot P(B \mid A)}{P(B)} \tag{4}$$

The ”naive” in Naïve Bayes comes from the assumption that all features are independent given the class [44]. That is:

$$P(x_1, x_2, \ldots, x_k \mid y) = P(x_1 \mid y) \cdot P(x_2 \mid y) \cdots P(x_k \mid y) \tag{5}$$

Thus, Bayes' theorem becomes:

$$P(y \mid x_1, \ldots, x_k) = \frac{P(y) \cdot \prod_{i=1}^{k} P(x_i \mid y)}{P(x_1)P(x_2) \cdots P(x_k)} \tag{6}$$

Since the denominator is constant for a given input, it can be written:

$$P(y \mid x_1, \ldots, x_k) \propto P(y) \cdot \prod_{i=1}^{k} P(x_i \mid y) \tag{7}$$

Naïve Bayes classifier is then constructed by computing the posterior for each class y and choosing the class with the highest probability:

$$\hat{y} = \arg\max_{y} P(y) \cdot \prod_{i=1}^{k} P(x_i \mid y) \tag{8}$$

Support Vector Machine is a supervised machine learning algorithm used for classification and regression tasks [29]. SVM finds the optimal hyperplane that separates data points belonging to different classes with the maximum margin, ensuring the model generalizes well to unseen data:

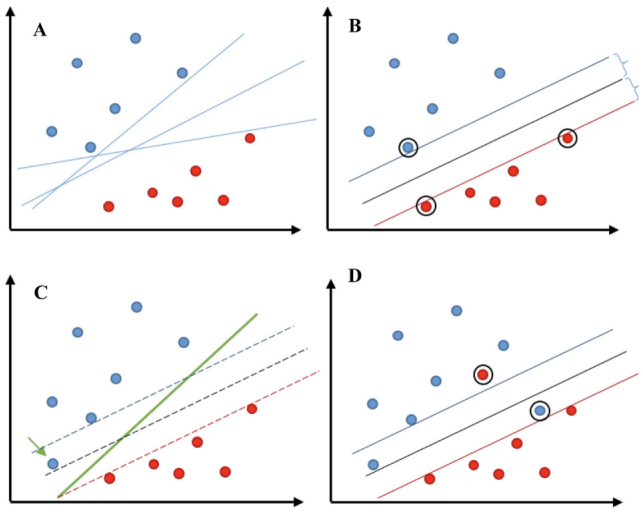

Figure 3. *A and B illustrate the principle of the maximum-margin classifier, C and D show the introduction of the slack variable to allow the support vector classifier to maximize its margin while disregarding the influence of nearby observations even when the data is inseparable [45].*

Mathematical formulation for SVM is as follows [32]:

- Decision Function:

  $f(x) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$, where **w** is the weight vector, **x** is the input feature vector, and b is the bias term. (9)

- Optimization problem for finding the optimal hyperplane:

  $\min_{\mathbf{w},b} \frac{1}{2}||\mathbf{w}||^2$, subject to: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \forall i$, where y_i is the class label ($+1$ or -1) for the i-th training example. (10)

- Optimization problem for soft-margin SVM:

  $\min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$, subject to: $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i$, where ξ_i are the slack variables (error allowances for each training example), C is the regularization parameter and n is the total number of training samples. (11)

Support Vector Machines can be divided into Linear SVM and Non-Linear SVM. Linear ones use a linear decision boundary to separate the data points of different classes. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line. It is done with the help of Kernel functions. For this thesis, since negative and positive sentiment were classified, Linear SVM were chosen.

To predict a continuous sentiment analysis value, Support Vector Regression (SVR) was used which involves adapting the principles of SVM to solve regression problems [30]. Instead of a binary classification (0 - negative, 1 - positive), continuous sentiment score between -1 and 1 is predicted.

Logistic Regression is a supervised machine learning algorithm used for classification problems where the model predicts the probability that a specific outcome occurs [21]. It works by modeling the relationship between the input features (words) and the class labels using a logistic (sigmoid) function [38]:

$$P(\text{Positive} \mid \text{document}) = \frac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}}, \text{ where} \quad (12)$$

- **x** is the vector of input features (the word vector for the document).
- **w** is the vector of weights associated with each feature.
- b is a bias term.
- The output is a probability between 0 and 1 that represents the likelihood of the document being positive.

Logistic Regression result of the classification is actually a value between [0, 1], which is interpreted as the probability $h(x)$ that the class of x is 1. The function used in the logistic regression is the logistic function, defined as [22]:

$$f(z) = \frac{1}{1 + e^{-z}}, \text{where} \quad (13)$$

$$z = \mathbf{w}^T \mathbf{x} + b \tag{14}$$

Recurrent Neural Network(RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step [34]. There are several variants of RNN [17]:

- Vanilla RNN - the most simple RNN, consists of a single hidden layer where weights are shared across time steps
- Bidirectional RNNs capture both past and future context for each time step.
- Long Short-Term Memory Networks (LSTMs) introduce a memory mechanism to overcome the vanishing gradient problem.
- Gated Recurrent Units (GRUs) - simplified LSTM, where the input and forget gates are combined into a single update gate.

Since it was decided to use only one deep learning approach in this thesis, Long Short-Term Memory Networks were chosen to represent this class of methods due to it's ability to perform complex tasks. LSTM architecture involves the memory cell which is controlled by three gates [26]:

1. Input Gate: Controls what information is added to the memory cell
2. Forget gate: Determines what information is removed from the memory cell
3. Output gate: Controls what information is output from the memory cell

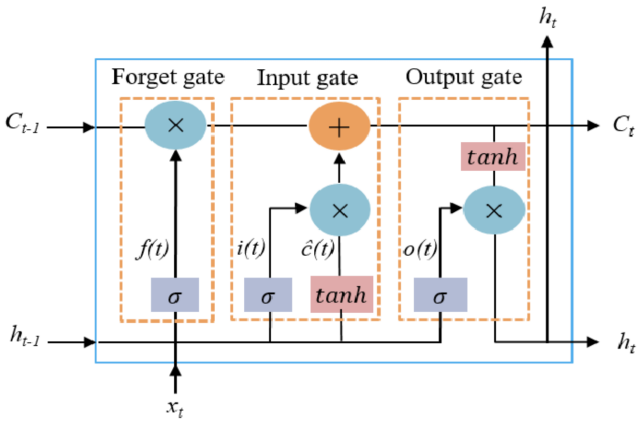

Figure 4. *Architecture of LSTM model [26].*

The equation for the forget gate is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f), \text{ where} \tag{15}$$

- W_f represents the weight matrix associated with the forget gate.
- $[h_{t-1}, x_t]$ denotes the concatenation of the current input and the previous hidden state.

- b_f is the bias with the forget gate.
- σ is the sigmoid activation function.

The equation for the input gate is:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \hat{C}_t &= tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ C_t &= f_t \odot C_{t-1} + i_t \odot \hat{C}_t, \text{ where} \end{aligned} \tag{16}$$

- $\odot$ denotes element-wise multiplication (Hadamard product)
- $tanh$ is the activation function

The equation for the output gate is:

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t \odot tanh(C_t), \text{ where} \end{aligned} \tag{17}$$

- o_t is the output gate activation.
- C_t is the current cell state.
- $\odot$ represents element-wise multiplication.
- σ is the sigmoid activation function.

2.3 Ensemble averaging

In order to improve the performance of sentiment analysis models, ensemble averaging was used as a final method to get better accuracy scores. Ensemble averaging utilizes results from various previous models and combines them to produce an even better predictive model. It operates by aggregating the predictions of diverse models, often achieved through averaging, weighted averaging, or combination techniques. Ensemble averaging addresses the bias-variance trade-off in machine learning. By combining diverse models, ensembles typically reduce variance and improve generalization.

2.4 Evaluation metrics and statistical tests

Several evaluation metrics were chosen to evaluate performance of the models:

- Classification model metrics:
  - Accuracy
  - Precision
  - Recall

  - F1 score

- Regression model metrics:
  - Mean Absolute Error
  - Mean Squared Error
  - Root Mean Squared Error
  - R^2 score

The output of the machine learning algorithm for classification tasks can be mapped to one of the four categories described in a confusion matrix [19]:

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 1. *Confusion Matrix*

Accuracy is the proportion of all classifications that were correct, whether positive or negative [8]. It is mathematically defined as:

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{total classifications}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{18}$$

While accuracy is often the first metric to be evaluated, it can be misleading in imbalanced datasets. In such scenarios, precision, recall, and F1 score provide deeper insights [19].

Precision is the proportion of all the model's positive classifications that are actually positive (measures the accuracy of positive predictions). It is mathematically defined as:

$$\text{Precision} = \frac{\text{correctly classified actual positives}}{\text{everything classified as positive}} = \frac{TP}{TP + FP} \tag{19}$$

Recall is a proportion of all actual positives that were classified correctly as positives (measures the model's ability to find all the positive instances):

$$\text{Recall} = \frac{\text{correctly classified actual positives}}{\text{all actual positives}} = \frac{TP}{TP + FN} \tag{20}$$

F1 score is the harmonic mean of precision and recall:

$$\text{F1 score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{21}$$

Unlike classification models, the performance of a regression model is reported as an error of model predictions and the metrics used for classification models cannot be used for regression tasks [6]. The Mean Absolute Error (MAE) is a frequently employed metric that measures the typical

absolute discrepancies between a dataset's actual values and projected values [7]. The formula to calculate MAE:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - y_i|, \text{ where} \tag{22}$$

- x_i represents the actual or observed values for the i-th data point.
- y_i represents the predicted value for the i-th data point.
- n represents number of observations.

Mean Squared Error (MSE) measures the square root of the average discrepancies between a dataset's actual values and projected values:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2, \text{ where} \tag{23}$$

- x_i represents the actual or observed value for the i-th data point.
- y_i represents the predicted value for the i-th data point.

Root Mean Squared Error (RMSE) is the square root of MSE and provides error magnitude in the same units as the original data, making interpretation more intuitive:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2} \tag{24}$$

R^2 score quantifies the percentage of the dependent variable's variation that the model's independent variables contribute to:

$$R^2 = 1 - \frac{SSR}{SST}, \text{ where} \tag{25}$$

- SSR represents the sum of squared residuals between the predicted values and actual values.
- SST represents the total sum of squares, which measures the total variance in the dependent variable.

To test for statistical significance of evaluation metrics between different model approaches, Friedman test is used in the analysis with the hypothesis:

$$H_0: \quad \text{Classification metrics are equal.} \tag{26}$$

$$H_1: \quad \text{Classification metrics are different.} \tag{27}$$

The statistic F_F is calculated as follows:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1) - \chi_F^2}, \text{where} \tag{28}$$

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k}R_j^2 - \frac{k(k+1)^2}{4}\right], \text{where} \tag{29}$$

- N is a number of blocks/observations (in our case - models)
- k is a number of conditions/treatments being compared (in our case - different combinations of training/testing datatsets)

F_F is compared with critical value $F_{k-1,(k-1)(N-1)}$ from Table 8. H_0 is rejected when F_F is greater than critical value.

If H_0 is rejected, Nemenyi post hoc test is performed to get the difference between mean rankings of the classification metrics. If this difference is greater than or equal to a CD (critic distance), two classifiers are significantly different from each other. CD is calculated as follows:

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{30}$$

The q_α (α = 0,05) term is obtained from Table 9.

3 Analysis

3.1 Datasets

In this thesis, five English-language sentiment analysis datasets were used:

- IMDB movie reviews (*link*) - it contains movie reviews in IMDB in a text format, together with the sentiment of each review. The dataset has around 50k movie reviews distributed into two sentiment classes - negative and positive.
- Rotten Tomatoes movie reviews (*link*) - contains more than 10k movie reviews in text format equally distributed between 2 sentiment classes (0 - negative, 1 - positive).
- Twitter (*link*) - contains data of 75k tweets on Twitter together with their sentiment in 3 classes (Positive, Negative and Neutral).
- Amazon product reviews (*link*) - contains almost 400k reviews on Amazon about different products on the platform together with a sentiment class (1 - negative, 2 - positive).
- Spotify app reviews (*link*) - contains more than 51k user reviews for Spotify from Google Play Store with each reviews labeled either positive or negative.

IMDB and Rotten Tomatoes datasets were taken as as example of datasets on the same domain (movies), but from different platforms. Amazon and Spotify datasets were taken to enhance the field by including more generalised reviews on various products. Twitter dataset was taken as a different type of dataset to see the behaviour of sentiment analysis models on completely unrelated field.

3.2 Data pre-processing

Before building sentiment analysis models, data cleaning and pre-processing techniques were applied to data. First of all, sentiment classes were aligned between the datasets, taking only negative and positive values (negative - 0, positive - 1). Then, all the pre-processing techniques described in section 2 were applied. For generalisation, datasets without domain-specific words were constructed for further parts of the analysis. Domain-specific words (e.g. ”actor”, ”scene” in movie reviews, ”music”, ”sound” in Spotify reviews, slang/curse words in Twitter tweets) were generated with the help from AI and removed from the text parts of the datasets. For this task, ChatGPT was used on the 8th of October 2025 and the full list of domain specific words can be found in Appendix 1.

The split for training/testing datasets was conducted as follows:

- For specific domain model evaluations (only one dataset is used), the 80/20 split was applied
- For cross-domain model evaluations (2 or more datasets used), one dataset acts as a testing one with one or more other datasets as training ones

Finally, feature extraction methods from section 2 were applied to training/testing datasets.

3.3 Model implementation and evaluation

A total of 13 different types of models, defined in Section 2.2 were implemented in the following way:

- Support Vector Machines, Logistic Regression, Naïve Bayes models and Support Vector Regression were implemented with each of the feature extraction techniques (BOW, TF-IDF, Word2Vec), resulting in a total of 12 different types of models per dataset.
- RNN (LSTM) with embedding layer instead of feature extraction and two LSTM layers trained for 3 epochs with a batch size of 60, which were chosen based on validation performance and computational constraints - small number of epochs is often enough to reach good validation performance while avoiding overfitting and batch size of 60 is large enough for reasonably stable gradient estimates, but small enough to fit GPU/CPU memory.

First of all, specific domain models (only one dataset used) were implemented and evaluated - 65 models (5 datasets each with 12 classic models + RNN) in total. Then, cross-domain models were implemented by taking one dataset as training one and another as testing (e.g. train - Amazon, test - IMDB). This was repeated for all the combinations of datasets and models - 260 models (20 dataset combinations each with 13 models) in total. Finally, domain-specific words were first removed from the training dataset and then from both training and testing datasets - 520 models (260 models repeated two times) in total. After implementing the models for all the combinations of 2 datasets, machine learning models (SVM, LR, NB) were run again with one dataset as testing one and four others combined into one as a training one to check whether expansion of the training dataset could enhance the metrics. Final approach implemented - ensemble averaging. It was only performed for the case where all 5 datasets are in use (one testing set and four others combined as training set) and for machine learning models only. Ensembles were grouped by feature extraction techniques(e.g. SVM(BOW), LR(BOW), NB(BOW) ensembled together). To implement all of the mentioned models, Python library `Scikit-learn` was used [36]. Because some of the models (especially SVR) were computationally heavy, VU HPC was used to run the SVR-specific portions of the code.

The resulting models were evaluated and their performance metrics are given in section 3.4.

3.4 Results

The evaluation metrics for each of the models were stored to later evaluate the differences. Results for the cases where IMDB was used as a testing set can be seen in Table 2 and Table 3. The results for the other datasets can be found in Appendix 2. Key findings are as follows:

- As expected, models trained on other datasets than they were tested on perform worse than the ones tested and trained on the same dataset.
- Models trained on Spotify and Amazon datasets perform better mostly due to more general datasets

- Models trained on Twitter dataset generally perform the worst.
- Cross-trained models between IMDB and Rotten Tomatoes do not perform as well as it could have been expected, meaning that data on the same domain differ a lot on different platforms.

Training set	Evaluation	SVR (BOW)	SVR (TF-IDF)	SVR (Word2Vec)
IMDB	MSE (RMSE)	0.01 (0.09)	0.01 (0.09)	0.01 (0.1)
	MAE	0.07	0.06	0.07
	R^2	0.73	0.76	0.69
Twitter	MSE (RMSE)	0.03 (0.16)	0.04 (0.19)	0.02 (0.15)
	MAE	0.13	0.15	0.12
	R^2	0.15	0.16	0.25
Tomatoes	MSE (RMSE)	**0.03 (0.17)**	0.02 (0.15)	**0.03 (0.17)**
	MAE	0.13	0.12	0.13
	R^2	0.08	0.29	0.01
Spotify	MSE (RMSE)	0.02 (0.15)	**0.04 (0.20)**	0.02 (0.14)
	MAE	0.11	0.16	0.11
	R^2	0.29	0.3	0.35
Amazon	MSE (RMSE)	**0.01 (0.12)**	**0.01 (0.11)**	**0.01 (0.10)**
	MAE	0.09	0.08	0.08
	R^2	0.54	0.62	0.65

Table 2. *Evaluation metrics for SVR (testing set - IMDB). Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Training Set	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
IMDB	Accuracy	0.8612	0.829	0.8614	0.8621	0.8325	0.8618	0.8525	0.8544	0.86
	F1 score	0.86; 0.86	0.83; 0.83	0.86; 0.86	0.86; 0.86	0.83; 0.84	0.86; 0.86	0.85; 0.86	0.85; 0.86	0.87
	Precision	0.87; 0.85	0.83; 0.83	0.87; 0.85	0.87; 0.86	0.84; 0.83	0.87; 0.85	0.86; 0.84	0.86; 0.84	0.86
	Recall	0.85; 0.88	0.82; 0.84	0.85; 0.87	0.85; 0.87	0.82; 0.85	0.85; 0.87	0.84; 0.87	0.84; 0.87	0.88
Twitter	Accuracy	**0.69278**	**0.64572**	**0.69418**	**0.70324**	**0.66226**	**0.69216**	0.66042	0.64396	**0.65**
	F1 score	0.66; 0.72	0.52; 0.72	0.65; 0.73	0.68; 0.72	0.57; 0.72	0.64; 0.73	0.68; 0.64	0.57; 0.69	0.69
	Precision	0.74; 0.66	0.80; 0.60	0.76; 0.66	0.73; 0.68	0.78; 0.61	0.77; 0.65	0.64; 0.68	0.71; 0.61	0.62
	Recall	0.59; 0.80	0.38; 0.91	0.57; 0.82	0.64; 0.76	0.45; 0.87	0.54; 0.84	0.73; 0.60	0.48; 0.81	0.79
Rotten Tomatoes	Accuracy	0.75834	0.7123	0.76236	0.76524	0.7089	0.75652	**0.60042**	**0.52484**	0.76
	F1 score	0.78; 0.73	0.77; 0.63	0.78; 0.74	0.78; 0.75	0.76; 0.62	0.78; 0.72	0.67; 0.49	0.63; 0.35	0.75
	Precision	0.72; 0.82	0.65; 0.89	0.72; 0.83	0.73; 0.81	0.64; 0.90	0.70; 0.85	0.57; 0.68	0.52; 0.55	0.77
	Recall	0.85; 0.66	0.94; 0.48	0.86; 0.66	0.83; 0.70	0.94; 0.47	0.89; 0.63	0.82; 0.38	0.80; 0.25	0.74
Spotify	Accuracy	0.73844	0.72036	0.74048	0.7267	0.68666	0.72306	0.70314	0.69796	0.73
	F1 score	0.75; 0.72	0.74; 0.69	0.75; 0.73	0.76; 0.68	0.74; 0.60	0.76; 0.68	0.73; 0.67	0.72; 0.67	0.71
	Precision	0.72; 0.77	0.69; 0.77	0.72; 0.77	0.68; 0.81	0.63; 0.83	0.68; 0.81	0.67; 0.75	0.67; 0.75	0.77
	Recall	0.79; 0.68	0.81; 0.63	0.79; 0.69	0.86; 0.59	0.90; 0.47	0.86; 0.59	0.79; 0.61	0.80; 0.60	0.66
Amazon	Accuracy	**0.83**	**0.821**	**0.83**	**0.832**	**0.822**	**0.831**	**0.811**	**0.811**	**0.83**
	F1 score	0.84; 0.82	0.83; 0.81	0.84; 0.82	0.84; 0.83	0.83; 0.82	0.84; 0.83	0.82; 0.80	0.82; 0.80	0.83
	Precision	0.81; 0.86	0.80; 0.85	0.80; 0.86	0.81; 0.86	0.81; 0.84	0.81; 0.86	0.77; 0.86	0.78; 0.86	0.84
	Recall	0.87; 0.79	0.86; 0.78	0.87; 0.79	0.87; 0.80	0.84; 0.80	0.87; 0.79	0.88; 0.75	0.88; 0.75	0.81

Table 3. *Evaluation metrics for machine learning models (testing set - IMDB). F1 score, precision, and recall are reported separately for each class: the first value corresponds to the negative class and the second to the positive class. Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Findings for each of the testing datasets:

- Testing set - IMDB:
  - In regards of cross-domain models, as it can be seen in Table 3, best results were achieved with Amazon dataset as training set, with the scores almost as high as domain models. All

evaluation metrics for these models were more or less consistent across with balanced class predictions. High R^2 and low error metrics suggest robust model performance. With Spotify and Rotten Tomatoes as training sets, a trend of class imbalance can be seen - models predict negative classes better than positive ones. Twitter as a training set performed the worst due to significant struggles to predict negative class. TF-IDF feature extraction generally resulted in more effective models with RNN model the one performing the most consistently in cross-domain.

- Testing set - Twitter:
  - Results in Table 11 show that all cross-domain models performed a lot worse than domain one, once again highlighting different nature of Twitter dataset. Models trained on IMDB data achieved more or less consistent modest accuracy (around 62%) and balanced class predictions with Word2Vec models performing slightly better. With Spotify and Rotten Tomatoes as training sets, once again a trend of bias towards negative class can be seen. Word2Vec models trained on Amazon data performed the best out of all cross-domain models with accuracy reaching 70% and better predictive class balance.
- Testing set - Rotten Tomatoes:
  - Lower scores for the domain model (especially with Word2Vec feature extraction) in Table 13 indicate that there might be some problems with the dataset in general. Cross-domain models trained on IMDB performed quite well, with Word2Vec models surpassing intra-domain models. With Spotify and Twitter as training sets, models struggled to predict positive class with Naïve Bayes and Logistic Regression models performing slightly better on it at the cost of negative class recall. Models trained on Amazon show moderate accuracy with quite balanced predictions for some of them, BOW models seem to struggle with negative class predictions, unlike Spotify and Twitter cross-domain models. In general, Word2Vec shows advantage over BOW and TF-IDF techniques when testing on Rotten Tomatoes data.
- Testing set - Spotify:
  - Surprisingly, results in Table 15 show that models trained on Twitter dataset performed the best out of cross-domain models with consistently high accuracy (near 80%) but with slight bias toward negative class in predictions. With Amazon and IMDB as training sets, models resulted in quite high accuracy scores with very high recall scores for positive class. Mixed R^2 values for IMDB models, from 0.33 to 0.59, highlight some predictive challenges nevertheless. Models with Rotten Tomatoes as training set performed slightly worse in comparison to others with struggles towards negative class. Higher regression error values reflect difficulty to strongly distinguish between sentiments.
- Testing set - Amazon:

– There is a noticeable gap in results in Table 17 for cross-domain models with IMDB and Spotify models performing better than Twitter or Rotten Tomatoes ones. Models trained on IMDB and Spotify showcase quite high accuracy scores with balanced class predictions. Low regression error values and quite high R^2 values suggest reasonable fit. Lower accuracy scores for models trained on Twitter and Rotten Tomatoes datasets indicate difficulties in capturing sentiments, especially for Word2Vec models.

Removal of domain-specific words did not improve the models as well as expected, with some training datasets and models benefiting from this kind of generalisation and some performing worse as the words removed had significant roles in sentiment detection. Evaluation metrics for models trained on Twitter and Amazon datasets and tested on IMDB before and after removing domain-specific words can be seen in Table 4 and Table 5. Rest of the tables are available in Appendix 3.

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Twitter	Not Removed	Accuracy	0.69278	0.64572	0.69418	0.70324	0.66226	0.69216	0.66042	0.64396	0.65
		F1 score	0.66; 0.72	0.52; 0.72	0.65; 0.73	0.68; 0.72	0.57; 0.72	0.64; 0.73	0.68; 0.64	0.57; 0.69	0.69
		Precision	0.74; 0.66	0.80; 0.60	0.76; 0.66	0.73; 0.68	0.78; 0.61	0.77; 0.65	0.64; 0.68	0.71; 0.61	0.62
		Recall	0.59; 0.80	0.38; 0.91	0.57; 0.82	0.64; 0.76	0.45; 0.87	0.54; 0.84	0.73; 0.60	0.48; 0.81	0.79
	Training set	Accuracy	0.68884	0.64306	0.69068	0.70148	0.65984	0.68982	0.65656	0.64912	0.66
		F1 score	0.65; 0.72	0.51; 0.72	0.65; 0.72	0.67; 0.73	0.56; 0.72	0.63; 0.73	0.68; 0.63	0.61; 0.68	0.67
		Precision	0.75; 0.65	0.81; 0.59	0.75; 0.65	0.74; 0.67	0.79; 0.61	0.78; 0.64	0.64; 0.68	0.69; 0.62	0.64
		Recall	0.57; 0.81	0.37; 0.91	0.57; 0.81	0.61; 0.79	0.43; 0.89	0.53; 0.85	0.73; 0.58	0.54; 0.76	0.69
	Training + Testing sets	Accuracy	0.7086	0.7066	0.71378	0.7163	0.71314	0.7164	0.65758	0.66536	0.66
		F1 score	0.70; 0.72	0.67; 0.74	0.70; 0.73	0.72; 0.71	0.70; 0.73	0.70; 0.73	0.70; 0.60	0.66; 0.67	0.68
		Precision	0.72; 0.70	0.77; 0.67	0.74; 0.69	0.71; 0.72	0.74; 0.69	0.75; 0.69	0.62; 0.72	0.68; 0.66	0.65
		Recall	0.68; 0.74	0.59; 0.82	0.65; 0.77	0.73; 0.71	0.66; 0.77	0.65; 0.78	0.80; 0.51	0.64; 0.69	0.72

Table 4. *Evaluation metrics for for ML models with domain-specific words removed (training set - Twitter, testing set - IMDB)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Amazon	Not removed	Accuracy	0.825	0.807	0.826	0.828	0.810	0.826	0.808	0.805	0.83
		F1 score	0.82; 0.83	0.80; 0.81	0.82; 0.83	0.82; 0.83	0.80; 0.82	0.82; 0.83	0.80; 0.81	0.80; 0.81	0.83
		Precision	0.84; 0.82	0.80; 0.81	0.83; 0.82	0.83; 0.82	0.82; 0.80	0.83; 0.82	0.80; 0.81	0.80; 0.81	0.84
		Recall	0.80; 0.85	0.80; 0.82	0.80; 0.85	0.81; 0.85	0.79; 0.83	0.81; 0.84	0.80; 0.81	0.80; 0.81	0.81
	Training set	Accuracy	0.829	0.818	0.830	0.832	0.820	0.830	0.814	0.811	0.83
		F1 score	0.84; 0.82	0.83; 0.81	0.84; 0.82	0.84; 0.83	0.82; 0.82	0.84; 0.82	0.82; 0.80	0.82; 0.80	0.82
		Precision	0.81; 0.86	0.79; 0.85	0.80; 0.86	0.81; 0.85	0.80; 0.84	0.81; 0.86	0.78; 0.86	0.77; 0.86	0.86
		Recall	0.87; 0.79	0.87; 0.77	0.87; 0.79	0.86; 0.80	0.84; 0.79	0.87; 0.79	0.87; 0.75	0.88; 0.74	0.79
	Training + Testing sets	Accuracy	0.832	0.823	0.832	0.833	0.821	0.832	0.816	0.817	0.83
		F1 score	0.83; 0.83	0.83; 0.82	0.84; 0.83	0.83; 0.83	0.82; 0.82	0.83; 0.83	0.82; 0.81	0.82; 0.81	0.83
		Precision	0.82; 0.84	0.81; 0.84	0.82; 0.85	0.83; 0.84	0.81; 0.83	0.82; 0.84	0.80; 0.84	0.80; 0.84	0.83
		Recall	0.85; 0.81	0.85; 0.80	0.85; 0.81	0.84; 0.82	0.83; 0.81	0.85; 0.81	0.85; 0.78	0.85; 0.78	0.83

Table 5. *Evaluation metrics for for ML models with domain-specific words removed (training set - Amazon, testing set - IMDB)*

It can be seen that results with IMDB as a testing set varies for different training sets. Domain-specific word removal worked quite well for models trained on Twitter and Amazon (when words

were removed both from training and testing sets), with all models resulting in slightly higher accuracy scores due to better predictions for negative values at the expense of positive ones. Models trained on Rotten Tomatoes and Spotify did not benefit as much from the removal of domain-specific words with several models performing slightly worse than the original variant.

With Twitter as a testing dataset, the models did not benefit from word removal indicating Twitter's more generic context.

Models tested on Rotten Tomatoes also showcase different behaviour to the removal of domain-specific words. Models built on Amazon, Twitter and Spotify benefit from this technique with recall for Twitter dataset improving a lot for negative values at the expense of positive ones. IMDB trained model performs very similarly before and after word removal, mostly due to the similarities of dataset domain.

Models built on Amazon, IMDB and Rotten Tomatoes trained on Spotify performed a lot better after removing domain-specific words with the exception of SVM models which performed similarly or slightly worse. With Twitter as a training set, models did not benefit from the technique, once again indicating Twitter's generic context.

Only models trained on Rotten Tomatoes performed better after word removal out of all models tested on Amazon due to better predictions for negative values at the expense of positive ones. All of the other dataset models performed very similarly before and after word removal.

Overall, it was observed that removal of domain-specific words resulted in quite different outcomes for different datasets and models, with models trained on Amazon benefitting most of the times and models trained or tested on Twitter performing worse or the same.

To compare if the evaluation metrics indicated significantly differ across different combinations of training/testing datasets and datasets with/without domain-specific words removed, Friedman test was used. The evaluation metrics that where found to be statistically different:

- IMDB as a testing set - Accuracy, Precision (only predictions for negative values, Recall (only predictions for positive values), F1 score
- Twitter as a testing set - Accuracy, Precision (only predictions for positive values), Recall, F1 score (only predictions for negative values)
- Rotten Tomatoes as a testing set - Accuracy
- Spotify as a testing set - Precision (only predictions for negative values), Recall (only predictions for positive values)
- Amazon as a testing set - Accuracy, Precision, Recall, F1 score

If Friedman test showed statistically significant difference, Nemenyi post hoc test was performed to find where exactly the statistically significant difference occurs. Some of the plots visualising the p-values of Nemenyi tests can be seen in Figure 5. All other plots can be found in Appendix 2. Training and testing dataset combinations are indicated (e.g. IMDB-Twitter means that IMDB was used as a training set and Twitter as testing set) together with the variant of domain-specific words removal (e.g. IMDB-Twitter means that all domain-specific words were retained, IMDB-Twitter_1 means that

words were removed from the training set and IMDB-Twitter_2 means that words were removed both from training and testing sets).

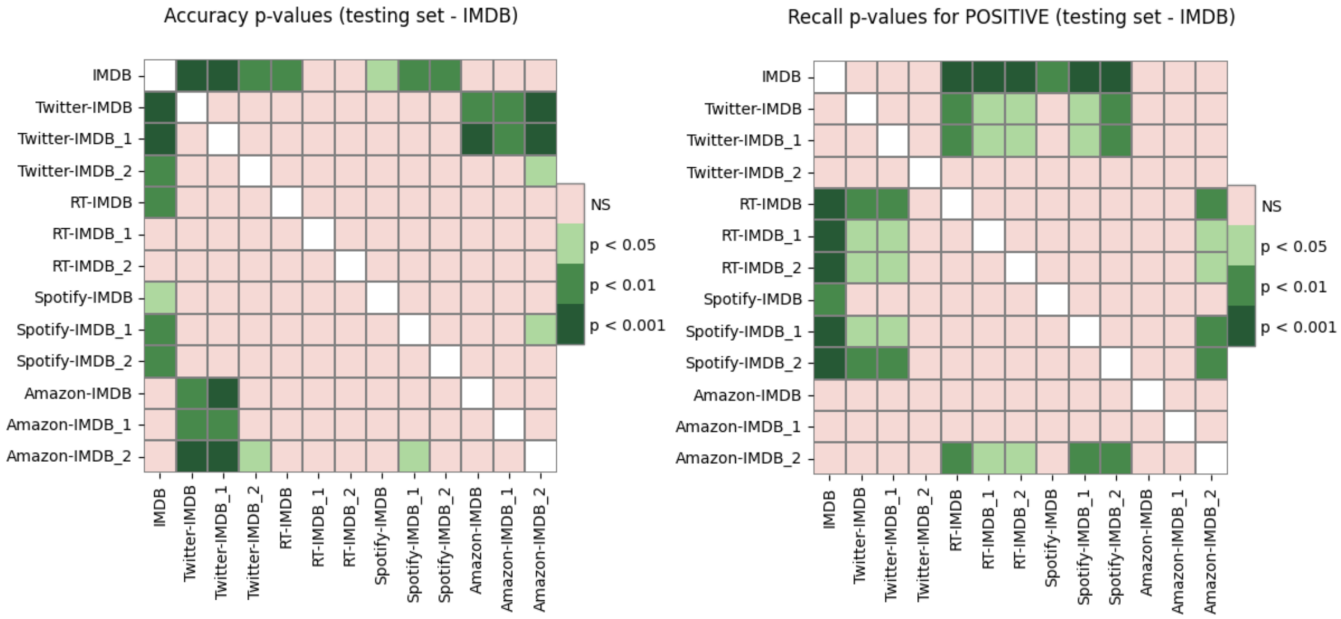

Figure 5. *Accuracy (left) and Recall (right) p-values*

With IMDB as a testing dataset, all of the models trained on other datasets performed worse and most of them performed statistically significantly worse, mean accuracy scores did not differ significantly only for models trained on Amazon and Rotten Tomatoes (with domain-specific words removed). Models trained on Twitter performed similarly to IMDB trained ones in terms of predicting positive values, however, predictions for negative values were much worse. Only Amazon trained models performed statistically indifferently from the original IMDB models.

IMDB and Rotten Tomatoes trained models performed statistically significantly worse on Twitter dataset than the original models. IMDB models lacked correct predictions for negative values, Rotten Tomatoes - for positive. Spotify and Amazon trained models mean metrics did not showcase statistically significant differences from Twitter models.

In terms of accuracy, none of the models had statistically significant differences when tested on Spotify data. The only significant differences came while predicting positive values (models trained on Rotten Tomatoes).

Twitter trained models were the only ones with the mean accuracy metrics significantly different from the original data models when tested on Rotten Tomatoes.

Rotten Tomatoes and Twitter trained models also were the only ones with statistically significant differences in mean accuracy scores when tested on Amazon data. Twitter models performed significantly worse with negative values predictions, Rotten Tomatoes - with positive. Overall, there were no statistically significant differences observed across the models before and after domain-specific words were removed, which indicates that this technique was not useful.

Another option for sentiment analysis evaluation across different domains that was tested in this thesis - taking one dataset as testing one and combining remaining ones into a single training set. Having a more generalised dataset for training did not improve the models as well as expected, with

most of the metrics taking values in between the ones that were evaluated with individual training sets. (Note - domain-specific words were retained in the datasets). The accuracies for ML models can be seen in Table 6:

Testing Set	Training set	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)
IMDB	IMDB	Accuracy	0.8612	0.829	0.8614	0.8621	0.8325	0.8618	0.8525	0.8544
IMDB	Remaining ones combined	Accuracy	0.8105	0.7621	0.8112	0.8074	0.7898	0.8067	0.795	0.7938
Twitter	Twitter	Accuracy	0.8022	0.7849	0.8021	0.8041	0.7884	0.8027	0.7548	0.7512
Twitter	Remaining ones combined	Accuracy	0.6685	0.6834	0.6738	0.6975	0.6891	0.6986	0.7213	0.7227
Rotten Tomatoes	Rotten Tomatoes	Accuracy	0.6958	0.7175	0.7104	0.7093	0.7099	0.7181	0.5633	0.5012
Rotten Tomatoes	Remaining ones combined	Accuracy	0.6395	0.6485	0.6428	0.65	0.647	0.6496	0.7216	0.7238
Spotify	Spotify	Accuracy	0.8776	0.8535	0.8779	0.8814	0.8581	0.8804	0.8633	0.8612
Spotify	Remaining ones combined	Accuracy	0.764	0.7074	0.766	0.769	0.7253	0.767	0.8058	0.8042
Amazon	Amazon	Accuracy	0.825	0.807	0.826	0.828	0.81	0.826	0.808	0.805
Amazon	Remaining ones combined	Accuracy	0.7808	0.7639	0.7829	0.7787	0.7653	0.7815	0.7881	0.7922

Table 6. *Accuracy metrics for ML models with singular and combined training sets*

As it can be observed, the accuracy scores for the models all went down in comparison to the models that were trained and tested on the same dataset. The difference between the scores is most noticeable in models tested on Twitter dataset, which once again suggests that Twitter dataset is very hard to predict without having the context of the tweets. Models tested on IMDB and Spotify have noticeable gaps in accuracy scores for the combined datasets, suggesting that these datasets are easier to predict without context than Twitter, but still lacks in performance without contextual data. Models tested on Amazon and Rotten Tomatoes datasets have lower accuracy scores for expanded training datasets, but the difference is much smaller comparing to other testing datasets, indicating that these datasets might have potential in generalisation of the sentiment analysis models.

Accuracy scores for the combined training set models were also tested with Friedman and Nemenyi tests to evaluate whether model performance declined statistically significantly or not. The results for testing sets with Friedman test H_0 rejected can be seen in Figure 6:

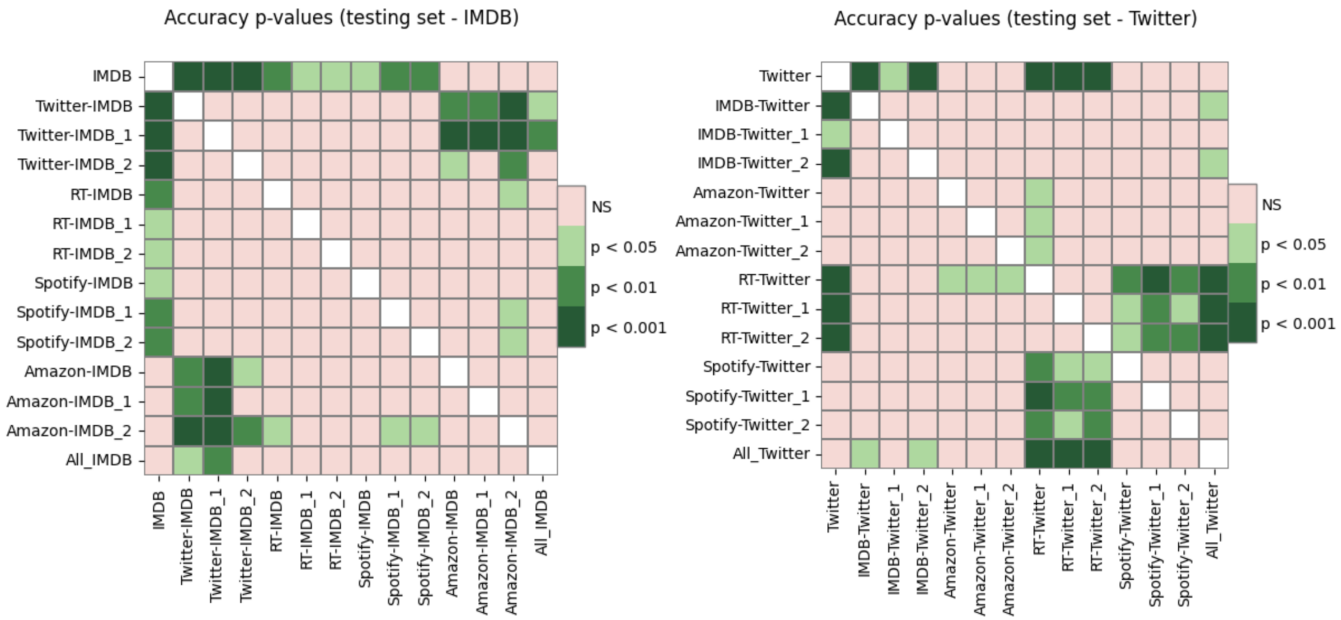

Figure 6. *Accuracy p-values for testing sets IMDB (left) and Twitter (right)*

All of the models with combined training sets performed worse than the original models, but the differences were not statistically significant. This indicates that generalisation and expansion of training datasets with different domain data can help with the performance of cross-domain sentiment analysis.

The last technique used in the analysis to improve model performance - ensemble averaging. Models trained only on combined training sets were taken and divided into different ensembles based on feature extraction techniques. The results are provided in Table 7.

Testing Set	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	Ensemble (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	Ensemble (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	Ensemble (Word2Vec)
IMDB	Accuracy	0.8105	0.7621	0.8112	0.8117	0.8074	0.7898	0.8067	0.8083	0.7949	0.7938	0.7932
IMDB	Precision	0.8579	0.6938	0.8566	0.8553	0.8564	0.7451	0.8559	0.8536	0.8403	0.8430	0.8443
IMDB	Recall	0.7442	0.9382	0.7474	0.7504	0.7388	0.8808	0.7375	0.7442	0.7282	0.7220	0.7190
IMDB	F1 score	0.7970	0.7977	0.7983	0.7994	0.7932	0.8073	0.7923	0.7951	0.7802	0.7778	0.7766
Twitter	Accuracy	0.6685	0.6834	0.6738	0.6729	0.6975	0.6891	0.6986	0.6990	0.7213	0.7227	0.7222
Twitter	Precision	0.6221	0.7215	0.6340	0.6323	0.7063	0.7334	0.7140	0.7131	0.7082	0.7123	0.7137
Twitter	Recall	0.7893	0.5551	0.7591	0.7620	0.6336	0.5542	0.6214	0.6246	0.7139	0.7089	0.7041
Twitter	F1 score	0.6958	0.6275	0.6909	0.6911	0.6680	0.6313	0.6645	0.6659	0.7111	0.7106	0.7089
Rotten Tomatoes	Accuracy	0.6395	0.6485	0.6428	0.6427	0.6501	0.6470	0.6496	0.6490	0.7216	0.7238	0.7233
Rotten Tomatoes	Precision	0.6224	0.6309	0.6248	0.6244	0.6566	0.6371	0.6537	0.6532	0.7015	0.7043	0.7063
Rotten Tomatoes	Recall	0.7093	0.7161	0.7149	0.7163	0.6293	0.6830	0.6363	0.6352	0.7714	0.7714	0.7646
Rotten Tomatoes	F1 score	0.6630	0.6708	0.6668	0.6672	0.6426	0.6593	0.6449	0.6441	0.7348	0.7363	0.7343
Spotify	Accuracy	0.7640	0.7074	0.7660	0.7643	0.7690	0.7253	0.7670	0.7672	0.8058	0.8042	0.8071
Spotify	Precision	0.6825	0.6191	0.6856	0.6830	0.6884	0.6379	0.6855	0.6860	0.7444	0.7421	0.7486
Spotify	Recall	0.8708	0.8770	0.8688	0.8703	0.8715	0.8746	0.8733	0.8723	0.8533	0.8533	0.8481
Spotify	F1 score	0.7652	0.7258	0.7664	0.7653	0.7692	0.7377	0.7681	0.7680	0.7951	0.7938	0.7953
Amazon	Accuracy	0.7808	0.7639	0.7829	0.7823	0.7787	0.7653	0.7815	0.7807	0.7881	0.7922	0.7917
Amazon	Precision	0.7571	0.7484	0.7651	0.7618	0.7754	0.7563	0.7810	0.7783	0.7748	0.7837	0.7847
Amazon	Recall	0.8442	0.8140	0.8332	0.8383	0.8013	0.8013	0.7985	0.8014	0.8283	0.8225	0.8193
Amazon	F1 score	0.7983	0.7798	0.7977	0.7982	0.7881	0.7781	0.7897	0.7897	0.8007	0.8027	0.8016

Table 7. *Evaluation metrics for ML models with and without ensemble averaging (training set - combined)*

TF-IDF and BOW ensembles of models tested on IMDB had higher accuracy scores, but their precision and recall metrics were lower, while Word2Vec ensemble accuracy was slighly lower than Logistic Regression and SVM models. Ensembles tested on Twitter data all performed a bit worse than sigular models with only TF-IDF ensemble accuracy slightly higher. Similar observations with ensembles tested on Rotten Tomatoes, only BOW ensemble recall and Word2Vec ensemble precision had slightly higher values. Only Word2Vec ensemble performed better in comparison to models tested on Spotify data and no ensembles tested on Amazon could reach the metrics of singular models. Overall, ensemble averaging did not enhance the sentiment analysis model performance, but the evaluation metrics are all very close to one another.

4 Conclusions

This thesis investigated the robustness and cross-domain generalisability of sentiment analysis models across five different datasets (IMDB, Rotten Tomatoes, Amazon, Spotify, Twitter), using different feature extraction methods (BOW, TF-IDF, Word2Vec), classical models (SVM, Logistic Regression, Naïve Bayes, SVR) and an LSTM based RNN. Overall, the models consistently performed best when trained and tested on the same dataset, confirming that sentiment is highly domain and platform dependent. Cross-domain performance dropped in all of the cases, but not uniformly: models trained on Amazon and Spotify, whose reviews are more generic and less domain bounded, transferred comparatively well and often approached intra-domain performance, whereas models trained on Twitter transferred the worst, reflecting the context dependent and informal nature of tweets. Surprisingly, even between two movie review datasets (IMDB and Rotten Tomatoes), cross-domain performance was weaker than expected, showing that topical similarity alone does not guarantee domain similarity.

At the model representation level, traditional machine learning methods with BOW and TF-IDF proved to be strong, reliable baselines, while Word2Vec representations offered advantages only for some dataset pairs (e.g. testing on Rotten Tomatoes) and were not universally superior. The LSTM model generally matched or slightly surpassed the best machine learning models, particularly in challenging cross-domain settings, but it did not eliminate the performance gap introduced by domain differences.

Attempts to improve cross-domain model performance through simple domain adaptation strategies produced mixed and often limited benefits. Removing domain-specific words did not improve performance and, in many cases, slightly harmed it, indicating that such words often carry genuinely useful sentiment information. Statistical testing (Friedman and Nemenyi tests) confirmed that there was no robust, general improvement pattern tied to word removal, and that differences before and after removal were mostly small and often not significant. Similarly, training on a large combined dataset did not outperform intra-domain models. Accuracy consistently decreased compared to models trained and tested on the same dataset. Statistical tests indicated that, although combined training models performed worse, the declines were not always statistically significant, so heterogeneous training data may still be acceptable when no intra-domain data are available, but they cannot fully replace domain-specific training. Finally, ensemble averaging per feature type yielded only marginal and inconsistent changes as in a few cases accuracy improved slightly, but ensembles rarely surpassed the best single model and never by a meaningful margin.

Taken together, the results show that domain mismatch is the dominant factor limiting sentiment analysis performance. Simple strategies such as removing domain-specific words, naive dataset aggregation, or basic ensembling provide, at best, modest gains. Future work should therefore consider more sophisticated domain adaptation or representation learning techniques, potentially supported by larger and more diverse training data and by deeper, more systematic use of AI tools when identifying domain-specific vocabulary.

References and sources

[1] A.AlQahtani. “Product Sentiment Analysis for Amazon Reviews.” In: *International Journal of Computer Science & Information Technology* 13 (2021), pages 15–30.

[2] H. Abubakarand, M. Umar. “Sentiment Classification: Review of Text Vectorization Methods: Bag of Words, Tf-Idf, Word2vec and Doc2vec.” In: *Sule Lamido University Journal of Science & Technology* 4 (2022), pages 27–33. URL: https://www.researchgate.net/publication/362825098_Sentiment_Classification_Review_of_Text_Vectorization_Methods_Bag_of_Words_Tf-Idf_Word2vec_and_Doc2vec.

[3] R. Ahuja, A. Chug, S.Kohli, S.Gupta, P.Ahuja. “The Impact of Features Extraction on the Sentiment Analysis.” In: *Procedia Computer Science* 152 (2019), pages 341–348.

[4] AIML.com. *What are the advantages and disadvantages of Bag-of-Words model?* URL: https://aiml.com/what-are-the-advantages-and-disadvantages-of-bag-of-words-model/ (viewed 2025-10-24).

[5] L. Z. B. Liu. *Mining Text Data*. 415–463 p. Chicago, United States of America: Springer, 2012.

[6] J. Brownlee. *Regression Metrics for Machine Learning*. URL: https://machinelearningmastery.com/regression-metrics-for-machine-learning/ (viewed 2025-10-30).

[7] D. Chicco, M. Warrens, G. Jurman. “The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation.” In: *PeerJ Computer Science* (2021). URL: https://peerj.com/articles/cs-623/.

[8] *Classification: Accuracy, recall, precision, and related metrics*. URL: https://developers.google.com/machine-learning/crash-course/classification/accuracy-precision-recall (viewed 2025-10-28).

[9] *Comparing classifiers (Friedman and Nemenyi tests)*. URL: https://medium.com/@diogeneswallis/comparing-classifiers-friedman-and-nemenyi-tests-32294103ee12 (viewed 2025-12-17).

[10] J. M. D. Jurafsky. *Speech and Language Processing*. 104 p. Pearson, 2025.

[11] D.M.W.Powers. “EVALUATION: FROM PRECISION, RECALL AND F-MEASURE TO ROC, INFORMEDNESS, MARKEDNESS & CORRELATION.” In: *Journal of Machine Learning Technologies* 2 (2011), pages 37–63.

[12] D. Donovan. *Understanding And Taking Advantage Of The Power Of Online Business Reviews*. URL: https://www.forbes.com/councils/forbesbusinesscouncil/2024/01/05/understanding-and-taking-advantage-of-the-power-of-online-business-reviews/ (viewed 2025-08-25).

[13] A. Downey. *Think Bayes*. 10 p. Sebastopol, CA, USA: O’Reilly, 2021. URL: https://books.google.lt/books?hl=en&lr=&id=Vh4vEAAAQBAJ&oi=fnd&pg=PR2&dq=Bayes%E2%80%99+Theorem&ots=HGbnAofJl9&sig=3qYz3KffpHLUsVCtlGxfNqGeam4&redir_esc=y#v=onepage&q&f=false.

[14] C. Y., R. Z., Y. L., L. J. *Survey of Specialized Large Language Model*. https://arxiv.org/pdf/2508.19667. 265 kB, viewed 2025-12-05. 2025.

[15] W. Yin, K. Kann, M. Yu, H. Schutze. *Comparative Study of CNN and RNN for Natural Language Processing*. https://arxiv.org/pdf/1702.01923. 283 kB, viewed 2025-09-05. 2017.

[16] *Installing NLTK Data*. URL: https://www.nltk.org/data.html (viewed 2025-11-03).

[17] W. Yu, I. Kim, C. Meschefske. "Analysis of different RNN autoencoder variants for time series classification and machine prognostics Author links open overlay panel." In: *Mechanical Systems and Signal Processing* 149 (2021). URL: https://www.sciencedirect.com/science/article/abs/pii/S0888327020307081.

[18] A. Jain. *TF-IDF in NLP (Term Frequency Inverse Document Frequency)*. URL: https://medium.com/@abhishekjainindore24/tf-idf-in-nlp-term-frequency-inverse-document-frequency-e05b65932f1d (viewed 2025-10-27).

[19] P. Kashyap. *Understanding Precision, Recall, and F1 Score Metrics*. URL: https://medium.com/@piyushkashyap045/understanding-precision-recall-and-f1-score-metrics-ea219b908093 (viewed 2025-10-28).

[20] E. Kavlakoglu. *What are Naïve Bayes classifiers*. URL: https://www.ibm.com/think/topics/naive-bayes (viewed 2025-10-27).

[21] F. Lee. *What is logistic regression?* URL: https://www.ibm.com/think/topics/logistic-regression (viewed 2025-10-27).

[22] M.Guerrero, J.Parada, H. Espitia. "EEG signal analysis using classification techniques: Logistic regression, artificial neural networks, support vector machines, and convolutional neural networks." In: *Heliyon* 7 (2021). URL: https://www.cell.com/heliyon/fulltext/S2405-8440(21)01361-X.

[23] M.R.Baker, Y.N.Taher, K.H.Jihad. "PREDICTION OF PEOPLE SENTIMENTS ON TWITTER USING MACHINE LEARNING CLASSIFIERS DURING RUSSIAN AGGRESSION IN UKRAINE." In: *Jordanian Journal of Computers and Information Technology* 9 (2023), pages 187–204.

[24] M.Siino, I.Tinnirellio, M.Cascia. "Is text preprocessing still worth the time? A comparative survey on the influence of popular preprocessing methods on Transformers and traditional classifiers Author links open overlay panel." In: *Information Systems* 121 (2024), pages 1–19.

[25] Y. Mao, Q. Liu, Y. Zhang. *Sentiment analysis methods, applications, and challenges: A systematic literature review*. https://www.sciencedirect.com/science/article/pii/S131915782400137X?ref=pdf_download&fr=RR-2&rr=97cdaa1f6e34c23e. 6.4 MB, viewed 2025-09-05. 2024.

[26] S. Mohsen, A. Elkaseer, S. Scholz. "Industry 4.0-Oriented Deep Learning Models for Human Activity Recognition." In: *IEEE Access* 9 (2021). URL: https://www.researchgate.net/publication/356018554_Industry_40-Oriented_Deep_Learning_Models_for_Human_Activity_Recognition.

[27] N.Chintalapudi, G.Battineni, M. Canio, G.G.Sagaroa, F.Amenta. “Text mining with sentiment analysis on seafarers’ medical documents.” In: *International Journal of Information Management Data Insights* (2021).

[28] *nltk.tokenize.punkt module*. URL: https://www.nltk.org/api/nltk.tokenize.punkt.html (viewed 2025-11-03).

[29] J. C. O. Lopez A. Lopez. *Multivariate Statistical Machine Learning Methods for Genomic Prediction*. 337–378 p. Springer, 2022.

[30] N. V. Otten. *Support Vector Regression (SVR) Simplified & How To Tutorial In Python*. URL: https://spotintelligence.com/2024/05/08/support-vector-regression-svr/ (viewed 2025-10-28).

[31] B. Pecher, I. Srba, M. Bielikova. *Comparing Specialised Small and General Large Language Models on Text Classification: 100 Labelled Samples to Achieve Break-Even Performance*. https://arxiv.org/abs/2402.12819. 594 kB, viewed 2025-10-24. 2024.

[32] V. Piccialli, M. Sciandrone. “Nonlinear optimization and support vector machines.” In: *Survey or Exposition* 314 (2022), pages 15–47.

[33] O. Rainio, J. Teuho, R. Klén. *Evaluation metrics and statistical tests for machine learning*. https://pmc.ncbi.nlm.nih.gov/articles/PMC10937649/pdf/41598_2024_Article_56706.pdf. 1.5 kB, viewed 2025-09-05. 2024.

[34] P. Raj. *Understanding Recurrent Neural Networks (RNN) — NLP*. URL: https://medium.com/@praveenraj.gowd/understanding-recurrent-neural-networks-rnn-nlp-e2f4cae03a4f (viewed 2025-10-27).

[35] X. Rong. *word2vec Parameter Learning Explained*. https://arxiv.org/abs/1411.2738. 817 kB, viewed 2025-12-05. 2016.

[36] *Scikit-learn: Machine Learning in Python*. URL: https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?source=post_page (viewed 2025-11-03).

[37] L. Si, D. Xhang, V. Rego. “Sentiment detection with auxiliary data.” In: *Discover Computing* 15 (2012), pages 373–390.

[38] R. Singh. *Natural Language Processing: Linear Text Classification*. URL: https://medium.com/@RobuRishabh/natural-language-processing-linear-text-classification-898f64a1e04a (viewed 2025-10-27).

[39] T.Singh, M.Kumari. “Role of Text Pre-processing in Twitter Sentiment Analysis.” In: *Procedia Computer Science* 89 (2016), pages 549–554.

[40] A. Tabassum, R. R. Patil. “A Survey on Text Pre-Processing & Feature Extraction Techniques in Natural Language Processing.” In: *IRJET Journal* (2020).

[41] Y. Tang, Y. Yang. *Do we need domain-specific embedding models? An empirical investigation*. https://arxiv.org/pdf/2409.18511. 907 kB, viewed 2025-10-24. 2024.

[42] J. Tao, X. Fang. “Toward multi-label sentiment analysis: a transfer learning based approach.” In: *Journal of Big Data* 7 (2020).

[43] E. Tonkin. *Working with Text*. 48 p. Cambridge, United Kingdom: Chandos Information Professional Series, 2016.

[44] V.Nakhipova, Y.Kerimbekov, Z. Umarova, L. Suleimenova, S. Botayeva, A. Ibashova, N. Zhumatayev. “Use of the Naive Bayes Classifier Algorithm in Machine Learning for Student Performance Prediction.” In: *International Journal of Information and Education Technology* 14 (2024), pages 92–98. URL: https://www.ijiet.org/vol14/IJIET-V14N1-2028.pdf.

[45] D. Valkenborg, A. Rousseau, M. Geubbelmans, T.Burzykowski. “Support vector machines.” In: *Statistics and research design* 164 (2023), pages 754–757. URL: https://www.ajodo.org/article/S0889-5406(23)00429-8/fulltext.

Appendix 1. AI usage

Artificial intelligence (AI) through ChatGPT was used in this thesis for two tasks:

- Python code cleaning - added comments to make the code more understandable and helped with some of the functions in the code (adjusted/rewrote my functions into cleaner versions using python libraries instead of manual calculations where applicable).
- Generated domain-specific words for each of the datasets used, datasets that were used for the analysis were not ingested to the prompt. The generated words are provided below.

Links to the exact ChatGPT prompts/conversations are not available because my company's internal ChatGPT instance was used, not the public site.

Domain-specific word list: Amazon

- product
- item
- goods
- purchase
- order
- package
- packaging
- shipment
- delivery
- shipping
- quality
- durability
- defect
- defective
- broken
- faulty
- damage
- warranty
- return
- refund
- feature
- features
- specifications
- size
- weight
- color
- design
- material
- performance
- functionality
- battery
- screen
- resolution
- sound
- speed
- customer
- service
- support
- helpful
- satisfied
- recommend
- recommendation
- disappointed
- expectation
- review
- reviews
- rating
- ratings
- feedback
- comment
- comments
- star
- stars
- price
- cost
- value
- expensive
- cheap
- affordable
- discount
- deal
- sale
- offer
- use
- used
- usage
- install
- installation
- compatible
- easy
- difficult
- manual
- fast
- slow
- delay
- tracking

- carrier
- electronics
- appliance
- clothing
- shoes
- book
- beauty
- toy
- game
- furniture
- tool

Domain-specific word list: Twitter

- rt
- retweet
- dm
- fomo
- idk
- lol
- lmao
- rofl
- brb
- omg
- wtf
- btw
- tbh
- imo
- imho
- irl
- smh
- afaik
- ftw
- ftl
- tl
- bf
- bff
- hashtag
- follow
- followers
- followback
- followfriday
- ff
- mention
- mentions
- like
- likes
- reply
- replies
- shit
- damn
- fuck
- fucking
- bitch
- bastard
- crap
- asshole
- dick
- piss
- bollocks
- bloody
- bugger
- arse
- cock
- cunt
- yo
- hey
- sup
- wanna
- gonna
- gotta
- kinda
- lemme
- ain’t
- cuz
- cause
- tho
- thx
- pls
- plz
- tweet
- tweets
- twitter
- timeline
- trending
- viral
- trends
- http
- https
- www
- com
- co
- twittercom
- tco

Domain-specific word list: Spotify

- app
- application
- spotify
- music
- song
- songs
- playlist
- playlists
- album
- albums
- artist
- artists

- stream
- streaming
- download
- downloads
- downloading
- downloaded
- shuffle
- repeat
- offline
- online
- library
- interface
- ui
- ux
- layout
- search
- searched
- searching
- recommendation
- recommendations
- suggest
- suggests
- suggested
- buffer
- buffering
- crash
- crashed
- crashing
- bug
- bugs
- glitch
- glitches
- premium
- subscription
- subscribe
- subscribed
- account
- login
- logged
- logging
- share
- shared
- sharing
- follow
- following
- followers

Domain-specific word list: IMDB and Rotten Tomatoes

- movie
- movies
- film
- films
- cinema
- screen
- plot
- story
- script
- screenplay
- scene
- scenes
- character
- characters
- role
- roles
- director
- directing
- direction
- actor
- actors
- actress
- actresses
- cast
- performance
- performances
- dialogue
- dialogues
- production
- producer
- studio
- editing
- editor
- set
- setting
- effect
- effects
- specialeffects
- cgi
- sound
- soundtrack
- music
- score
- cinematography
- lighting
- camera
- angle
- review
- reviews
- rating
- ratings
- critique
- critic
- critics
- award
- awards
- nomination
- nominations
- festival
- screening
- comedy
- drama
- thriller
- horror
- romance
- action
- adventure
- sci-fi

- sciencefiction
- fantasy
- documentary
- animation
- biography
- crime
- classic
- blockbuster
- release
- released
- premiere
- premiered
- boxoffice
- gross
- budget
- trailer
- teaser

Appendix 2. Figures

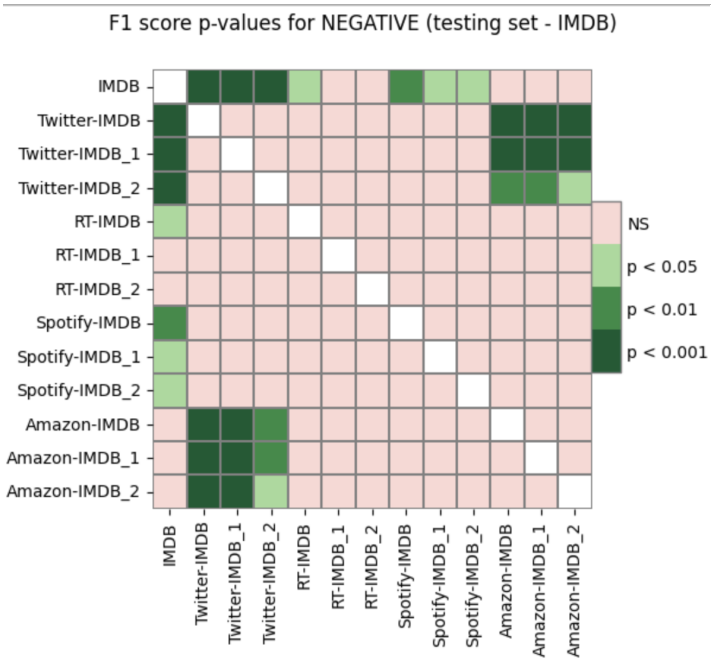

Figure 7. *F1 score p-values (IMDB)*

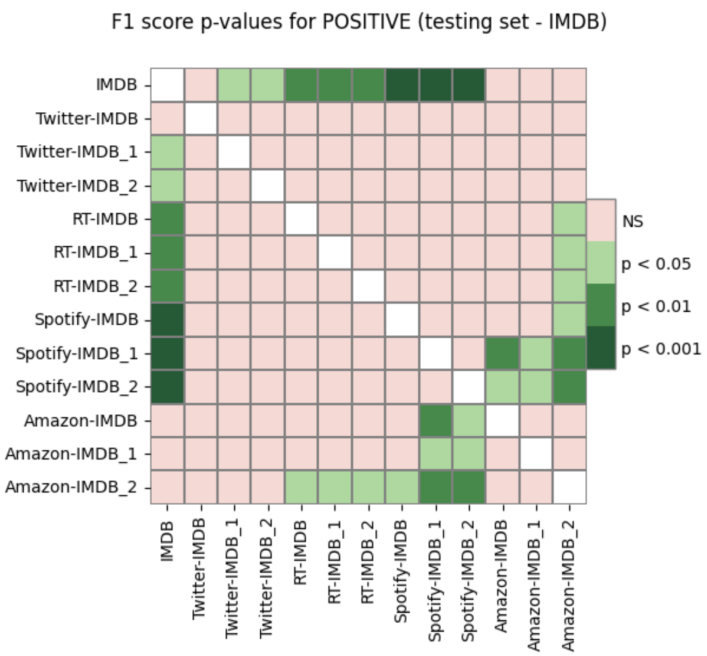

Figure 8. *F1 score p-values (IMDB)*

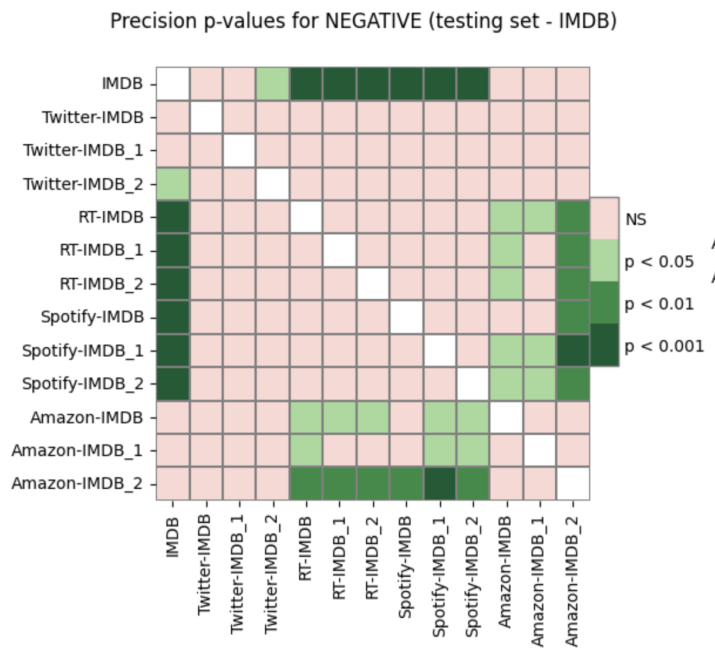

Figure 9. *Precision p-values (IMDB)*

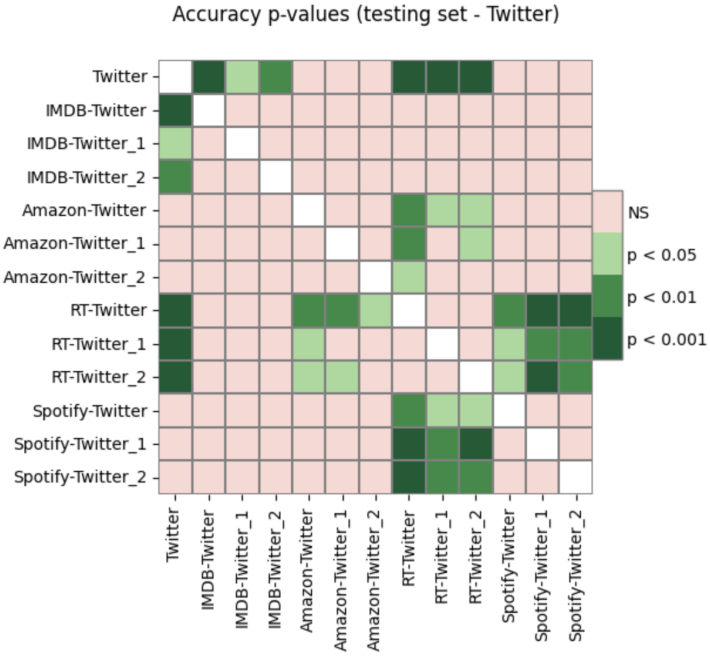

Figure 10. *Accuracy p-values (Twitter)*

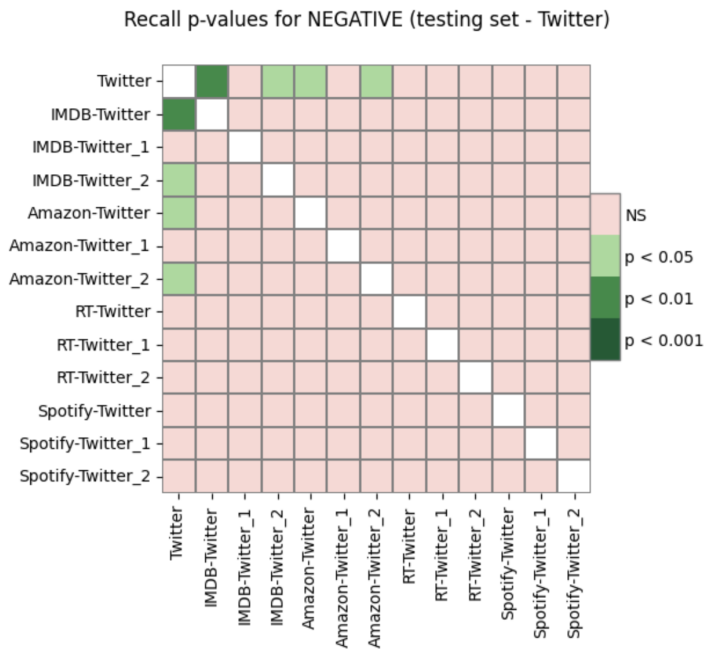

Figure 11. *Recall p-values (Twitter)*

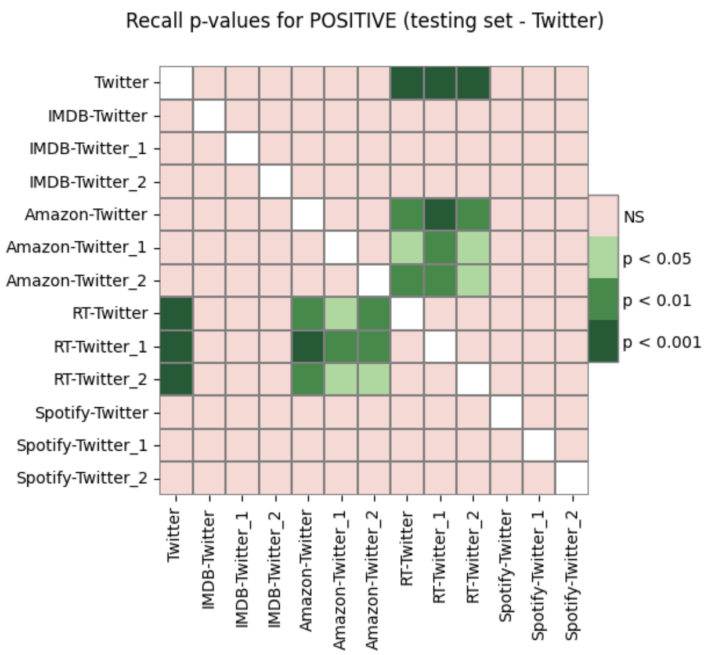

Figure 12. *Recall p-values (Twitter)*

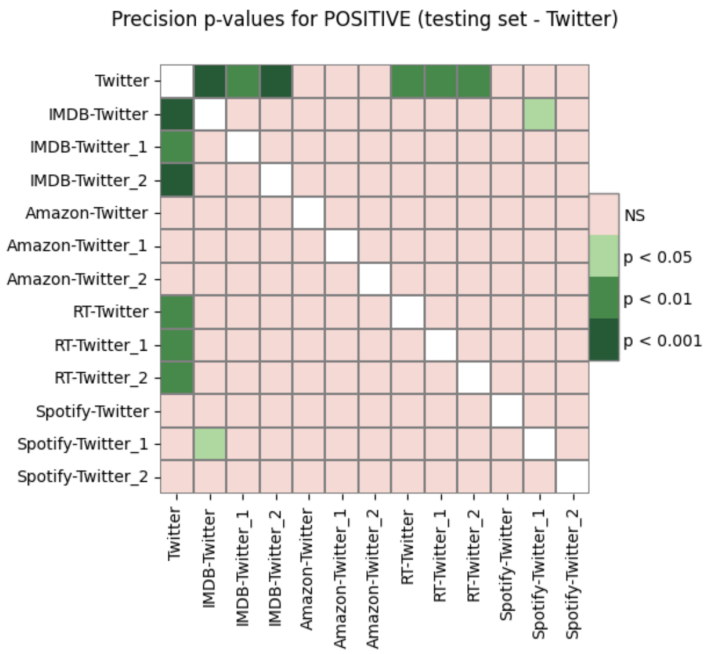

Figure 13. *Precision p-values (Twitter)*

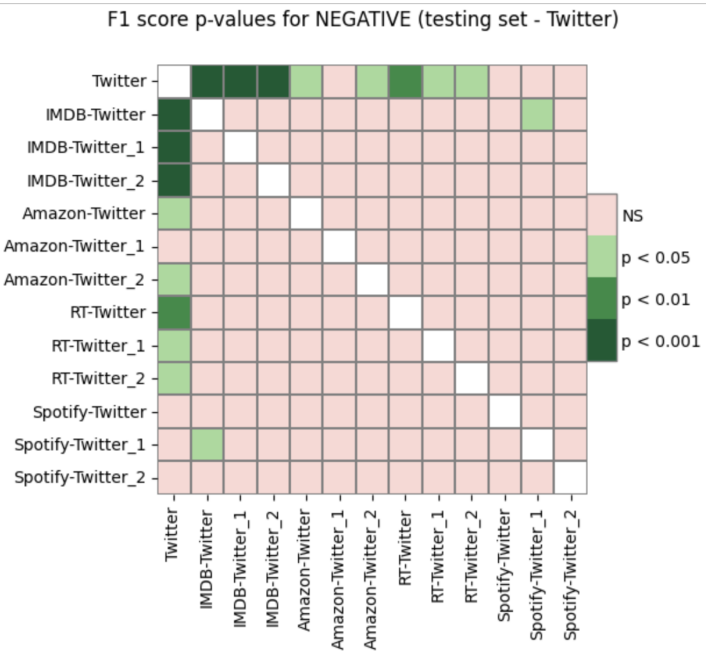

Figure 14. *F1 score p-values (Twitter)*

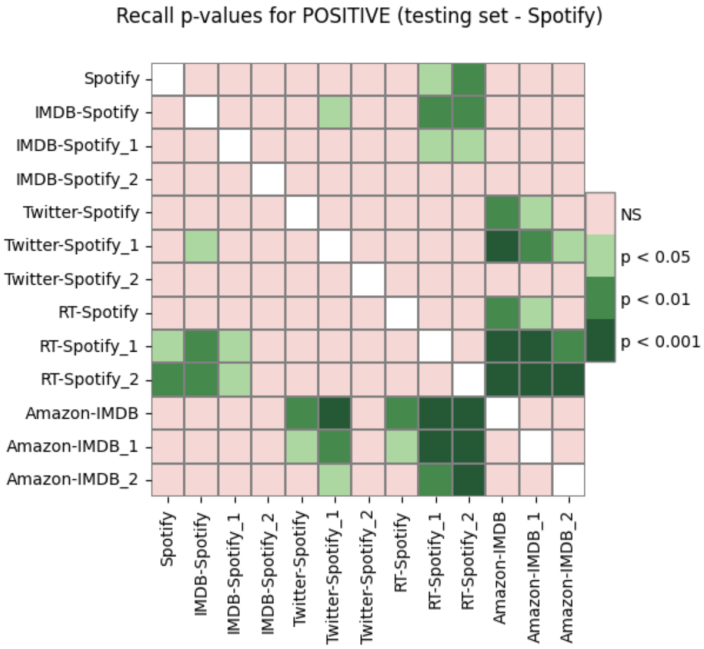

Figure 15. *Recall p-values (Spotify)*

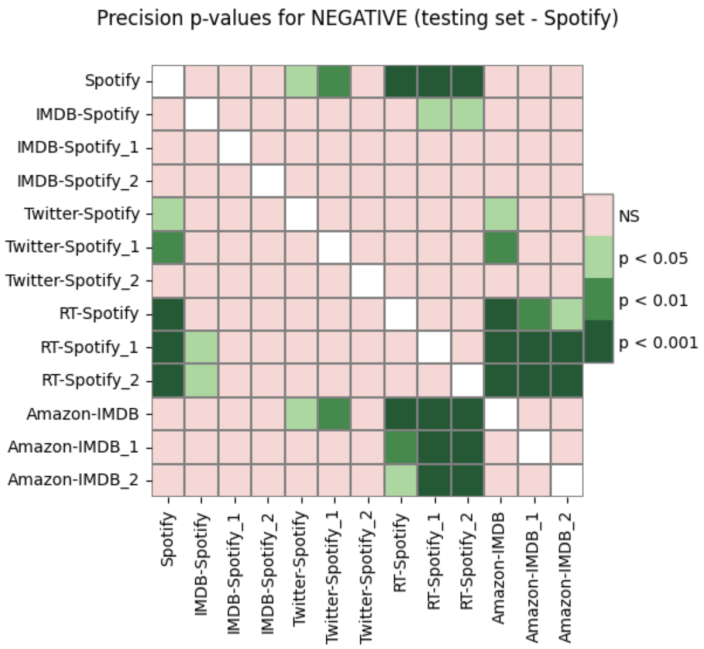

Figure 16. *Precision p-values (Spotify)*

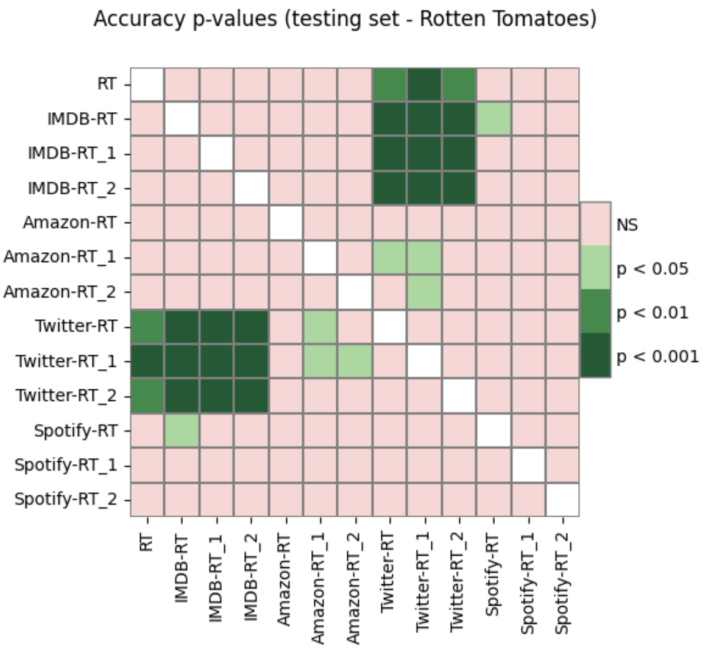

Figure 17. *Accuracy p-values (Rotten Tomatoes)*

Figure 18. *Accuracy p-values (Amazon)*

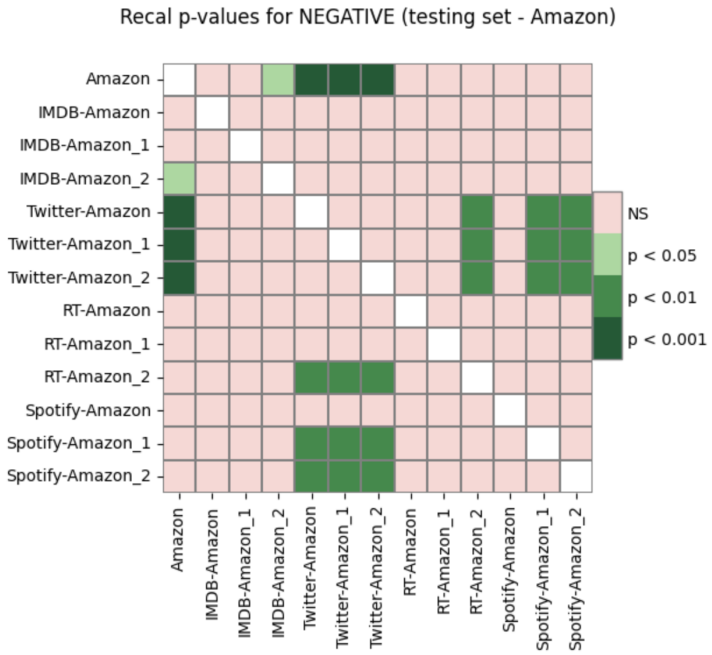

Figure 19. *Recall p-values (Amazon)*

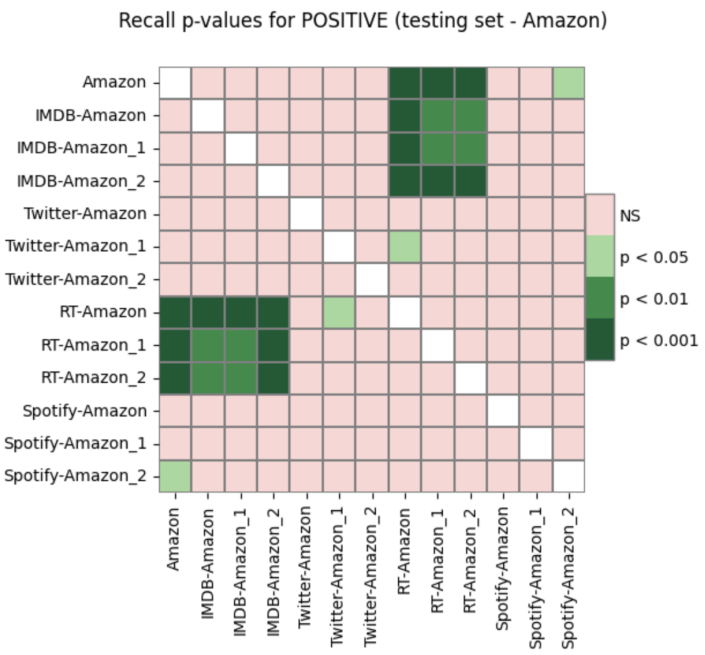

Figure 20. *Recall p-values (Amazon)*

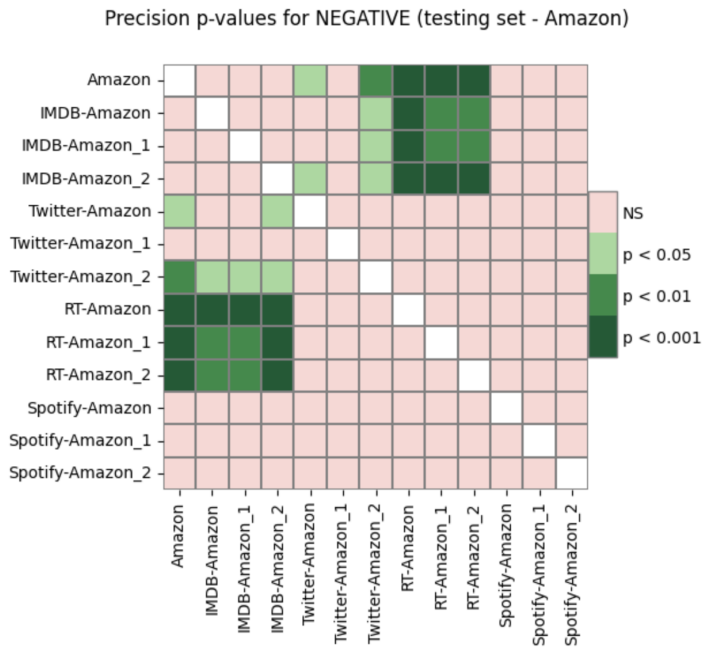

Figure 21. *Precision p-values (Amazon)*

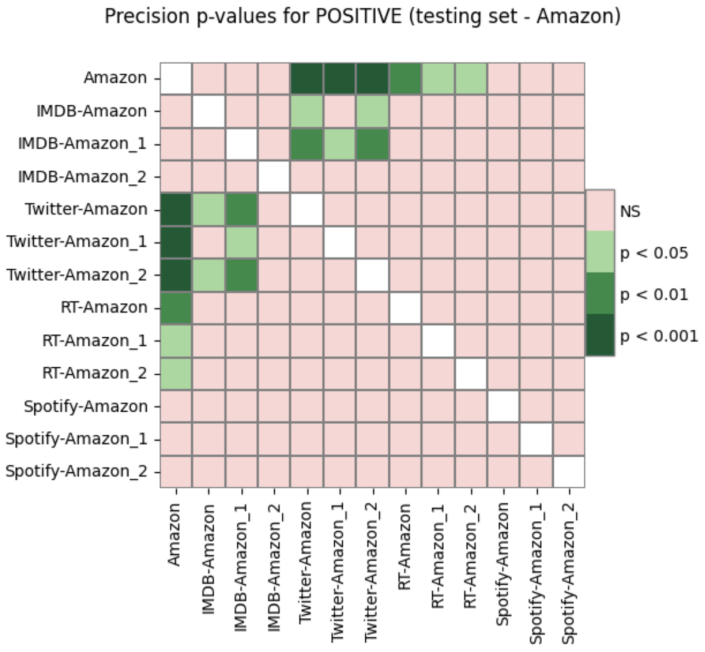

Figure 22. *Precision p-values (Amazon)*

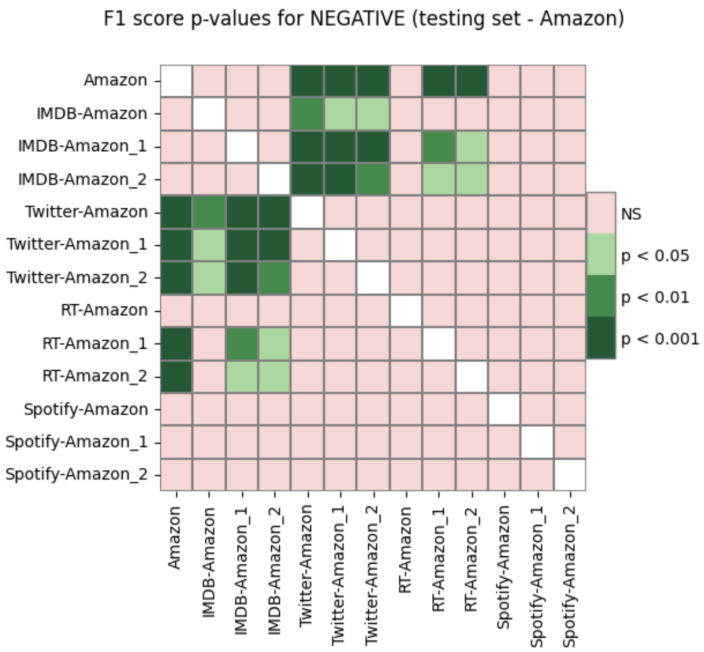

Figure 23. *F1 score p-values (Amazon)*

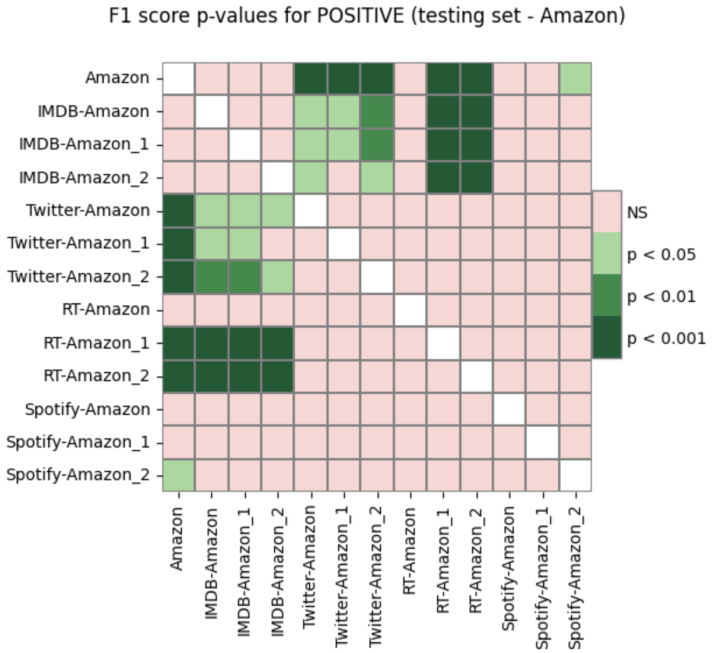

Figure 24. *F1 score p-values (Amazon)*

Appendix 3. Tables

	$df_1 = 1$	2	3	4	5	6	7	8	9	10	12	15	20
$df_2 = 1$	161.4476	199.5000	215.7073	224.5832	230.1619	233.9860	236.7684	238.8827	240.5433	241.8817	243.9060	245.9499	248.0131
2	18.5128	19.0000	19.1643	19.2468	19.2964	19.3295	19.3532	19.3710	19.3848	19.3959	19.4125	19.4291	19.4458
3	10.1280	9.5521	9.2766	9.1172	9.0135	8.9406	8.8867	8.8452	8.8123	8.7855	8.7446	8.7029	8.6602
4	7.7086	6.9443	6.5914	6.3882	6.2561	6.1631	6.0942	6.0410	5.9988	5.9644	5.9117	5.8578	5.8025
5	6.6079	5.7861	5.4095	5.1922	5.0503	4.9503	4.8759	4.8183	4.7725	4.7351	4.6777	4.6188	4.5581
6	5.9874	5.1433	4.7571	4.5337	4.3874	4.2839	4.2067	4.1468	4.0990	4.0600	3.9999	3.9381	3.8742
7	5.5914	4.7374	4.3468	4.1203	3.9715	3.8660	3.7870	3.7257	3.6767	3.6365	3.5747	3.5107	3.4445
8	5.3177	4.4590	4.0662	3.8379	3.6875	3.5806	3.5005	3.4381	3.3881	3.3472	3.2839	3.2184	3.1503
9	5.1174	4.2565	3.8625	3.6331	3.4817	3.3738	3.2927	3.2296	3.1789	3.1373	3.0729	3.0061	2.9365
10	4.9646	4.1028	3.7083	3.4780	3.3258	3.2172	3.1355	3.0717	3.0204	2.9782	2.9130	2.8450	2.7740
11	4.8443	3.9823	3.5874	3.3567	3.2039	3.0946	3.0123	2.9480	2.8962	2.8536	2.7876	2.7186	2.6464
12	4.7472	3.8853	3.4903	3.2592	3.1059	2.9961	2.9134	2.8486	2.7964	2.7534	2.6866	2.6169	2.5436
13	4.6672	3.8056	3.4105	3.1791	3.0254	2.9153	2.8321	2.7669	2.7144	2.6710	2.6037	2.5331	2.4589
14	4.6001	3.7389	3.3439	3.1122	2.9582	2.8477	2.7642	2.6987	2.6458	2.6022	2.5342	2.4630	2.3879
15	4.5431	3.6823	3.2874	3.0556	2.9013	2.7905	2.7066	2.6408	2.5876	2.5437	2.4753	2.4034	2.3275

Table 8. *F critical values for $\alpha = 0.05$ [9]. df_1 corresponds to $k - 1$, df_2 to $(k - 1)(N - 1)$*

k	2	3	4	5	6	7	8	9	10
Nemenyi	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164

Table 9. *Nemenyi critical values [9]*

Training set	Evaluation	SVR (BOW)	SVR (TF-IDF)	SVR (Word2Vec)
Twitter	MSE (RMSE)	0.02 (0.15)	0.02 (0.15)	0.07 (0.27)
	MAE	0.10	0.10	0.19
	R^2	0.83	0.83	0.46
IMDB	MSE (RMSE)	0.10 (0.31)	0.08 (0.29)	0.08 (0.29)
	MAE	0.23	0.21	0.21
	R^2	0.24	0.35	0.36
Rotten Tomatoes	MSE (RMSE)	0.07 (0.27)	0.07 (0.27)	0.12 (0.35)
	MAE	0.19	0.19	0.26
	R^2	0.45	0.44	0.03
Spotify	MSE (RMSE)	**0.06 (0.24)**	**0.06 (0.24)**	0.08 (0.29)
	MAE	0.17	0.17	0.21
	R^2	0.54	0.54	0.36
Amazon	MSE (RMSE)	0.07 (0.26)	**0.06 (0.24)**	**0.07 (0.27)**
	MAE	0.19	0.17	0.19
	R^2	0.46	0.55	0.45

Table 10. *Evaluation metrics for SVR (testing set - Twitter). Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Training Set	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2Vec)	LR (Word2Vec)	RNN
Twitter	Accuracy	0.8022	0.7849	0.8021	0.8041	0.7884	0.8027	0.7548	0.7512	0.92
	F1 score	0.81; 0.80	0.80; 0.77	0.81; 0.79	0.81; 0.80	0.81; 0.77	0.81; 0.79	0.77; 0.74	0.76; 0.74	0.91
	Precision	0.83; 0.78	0.78; 0.79	0.82; 0.78	0.82; 0.78	0.78; 0.80	0.82; 0.78	0.77; 0.74	0.76; 0.74	0.90
	Recall	0.79; 0.81	0.82; 0.75	0.79; 0.81	0.80; 0.81	0.83; 0.74	0.80; 0.81	0.76; 0.74	0.76; 0.74	0.93
IMDB	Accuracy	0.621	0.628	0.625	0.626	0.624	0.629	0.67	0.67	0.62
	F1 score	0.64; 0.61	0.65; 0.60	0.65; 0.60	0.64; 0.61	0.65; 0.60	0.64; 0.61	0.67; 0.67	0.67; 0.67	0.68
	Precision	0.64; 0.61	0.63; 0.62	0.63; 0.62	0.64; 0.61	0.63; 0.62	0.64; 0.61	0.69; 0.65	0.69; 0.65	0.57
	Recall	0.63; 0.61	0.68; 0.57	0.66; 0.58	0.64; 0.61	0.67; 0.58	0.65; 0.61	0.65; 0.69	0.65; 0.69	0.83
Rotten Tomatoes	Accuracy	**0.591**	**0.598**	**0.594**	**0.597**	**0.60**	**0.603**	**0.526**	**0.453**	**0.57**
	F1 score	0.66; 0.48	0.67; 0.49	0.66; 0.49	0.66; 0.49	0.67; 0.49	0.67; 0.51	0.55; 0.50	0.36; 0.52	0.63
	Precision	0.58; 0.62	0.58; 0.63	0.58; 0.62	0.59; 0.62	0.59; 0.64	0.59; 0.63	0.54; 0.51	0.46; 0.45	0.54
	Recall	0.77; 0.40	0.78; 0.40	0.76; 0.41	0.77; 0.41	0.79; 0.39	0.76; 0.43	0.55; 0.50	0.30; 0.62	0.76
Spotify	Accuracy	**0.689**	**0.6902**	**0.688**	**0.689**	**0.679**	**0.693**	0.67	0.669	**0.70**
	F1 score	0.68; 0.70	0.74; 0.62	0.67; 0.70	0.69; 0.69	0.74; 0.58	0.70; 0.69	0.70; 0.64	0.70; 0.63	0.68
	Precision	0.74; 0.65	0.66; 0.76	0.74; 0.65	0.72; 0.66	0.64; 0.78	0.71; 0.67	0.67; 0.68	0.66; 0.68	0.70
	Recall	0.62; 0.76	0.85; 0.52	0.62; 0.76	0.66; 0.73	0.88; 0.46	0.69; 0.70	0.74; 0.60	0.74; 0.59	0.66
Amazon	Accuracy	0.638	0.666	0.646	0.679	0.662	0.681	**0.702**	**0.705**	0.68
	F1 score	0.57; 0.69	0.66; 0.68	0.60; 0.68	0.69; 0.66	0.64; 0.68	0.70; 0.66	0.72; 0.68	0.73; 0.68	0.69
	Precision	0.74; 0.59	0.71; 0.63	0.73; 0.60	0.69; 0.67	0.71; 0.62	0.69; 0.67	0.70; 0.70	0.70; 0.71	0.65
	Recall	0.47; 0.83	0.61; 0.73	0.50; 0.80	0.70; 0.66	0.59; 0.74	0.70; 0.66	0.75; 0.65	0.75; 0.65	0.73

Table 11. *Evaluation metrics for machine learning models (testing set - Twitter). Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Training set	Evaluation	SVR (BOW)	SVR (TF-IDF)	SVR (Word2Vec)
Rotten Tomatoes	MSE (RMSE)	0.05 (0.23)	0.05 (0.23)	0.09 (0.30)
	MAE	0.17	0.17	0.22
	R^2	0.43	0.42	0.05
IMDB	MSE (RMSE)	**0.07 (0.26)**	0.06 (0.25)	**0.06 (0.24)**
	MAE	0.20	0.18	0.18
	R^2	0.23	0.32	0.33
Twitter	MSE (RMSE)	0.06 (0.25)	0.06 (0.25)	**0.08 (0.28)**
	MAE	0.18	0.18	0.21
	R^2	0.29	0.28	0.14
Spotify	MSE (RMSE)	0.06 (0.25)	0.06 (0.25)	0.07 (0.27)
	MAE	0.19	0.19	0.20
	R^2	0.28	0.28	0.20
Amazon	MSE (RMSE)	**0.06 (0.24)**	**0.06 (0.24)**	0.06 (0.25)
	MAE	0.18	0.18	0.19
	R^2	0.34	0.34	0.32

Table 12. *Evaluation metrics for SVR (testing set - Rotten Tomatoes). Best performing cross-domain models in **green**, worst - in **red**.*

Training Set	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2Vec)	LR (Word2Vec)	RNN
Rotten Tomatoes	Accuracy	0.6958	0.7175	0.7104	0.7093	0.7099	0.7181	0.5633	0.5012	0.74
	F1 score	0.71; 0.68	0.72; 0.71	0.72; 0.70	0.72; 0.70	0.72; 0.70	0.72; 0.71	0.57; 0.56	0.53; 0.47	0.75
	Precision	0.68; 0.72	0.71; 0.73	0.69; 0.73	0.69; 0.73	0.70; 0.72	0.71; 0.73	0.56; 0.57	0.50; 0.50	0.72
	Recall	0.75; 0.64	0.73; 0.70	0.75; 0.67	0.75; 0.67	0.73; 0.69	0.74; 0.70	0.58; 0.58	0.56; 0.44	0.78
IMDB	Accuracy	**0.645**	**0.655**	**0.647**	**0.65**	**0.657**	**0.651**	**0.717**	**0.717**	0.67
	F1 score	0.64; 0.65	0.63; 0.68	0.65; 0.64	0.65; 0.65	0.63; 0.68	0.65; 0.65	0.70; 0.73	0.70; 0.73	0.7
	Precision	0.65; 0.64	0.68; 0.64	0.64; 0.65	0.65; 0.65	0.68; 0.64	0.65; 0.65	0.75; 0.69	0.75; 0.69	0.65
	Recall	0.63; 0.66	0.59; 0.72	0.67; 0.63	0.65; 0.65	0.59; 0.72	0.64; 0.66	0.66; 0.77	0.66; 0.77	0.75
Twitter	Accuracy	**0.548**	**0.553**	**0.559**	**0.552**	**0.553**	**0.562**	**0.536**	**0.549**	**0.57**
	F1 score	0.62; 0.44	0.54; 0.57	0.50; 0.60	0.58; 0.51	0.55; 0.56	0.51; 0.61	0.58; 0.49	0.54; 0.56	0.61
	Precision	0.53; 0.58	0.56; 0.55	0.58; 0.55	0.54; 0.56	0.55; 0.55	0.58; 0.55	0.53; 0.55	0.55; 0.55	0.55
	Recall	0.74; 0.35	0.52; 0.58	0.45; 0.67	0.63; 0.47	0.54; 0.56	0.45; 0.68	0.64; 0.44	0.53; 0.56	0.68
Spotify	Accuracy	0.597	0.583	0.597	0.605	0.584	0.604	0.588	0.587	0.64
	F1 score	0.53; 0.64	0.64; 0.51	0.53; 0.65	0.57; 0.64	0.66; 0.47	0.59; 0.62	0.60; 0.58	0.60; 0.57	0.62
	Precision	0.63; 0.58	0.56; 0.62	0.63; 0.58	0.63; 0.59	0.56; 0.65	0.61; 0.60	0.58; 0.59	0.58; 0.59	0.65
	Recall	0.46; 0.73	0.76; 0.43	0.46; 0.73	0.52; 0.69	0.80; 0.37	0.56; 0.65	0.61; 0.57	0.62; 0.55	0.59
Amazon	Accuracy	0.628	0.64	0.63	0.632	0.638	0.636	0.676	0.676	**0.69**
	F1 score	0.56; 0.68	0.61; 0.67	0.58; 0.67	0.64; 0.63	0.60; 0.67	0.64; 0.63	0.67; 0.68	0.67; 0.68	0.7
	Precision	0.69; 0.60	0.66; 0.62	0.67; 0.60	0.63; 0.63	0.67; 0.62	0.63; 0.64	0.68; 0.67	0.68; 0.67	0.68
	Recall	0.47; 0.79	0.56; 0.72	0.51; 0.75	0.64; 0.62	0.54; 0.74	0.65; 0.62	0.67; 0.69	0.67; 0.68	0.72

Table 13. *Evaluation metrics for machine learning models (testing set - Rotten Tomatoes). Best performing cross-domain models in **green**, worst - in **red**.*

Training set	Evaluation	SVR (BOW)	SVR (TF-IDF)	SVR (Word2Vec)
Spotify	MSE (RMSE)	0.03 (0.17)	0.03 (0.16)	0.05 (0.22)
	MAE	0.12	0.12	0.15
	R^2	0.80	0.81	0.66
IMDB	MSE (RMSE)	**0.09 (0.30)**	0.06 (0.24)	0.09 (0.30)
	MAE	0.23	0.18	0.22
	R^2	0.33	0.59	0.35
Twitter	MSE (RMSE)	**0.05 (0.22)**	0.04 (0.21)	0.07 (0.27)
	MAE	0.16	0.15	0.20
	R^2	0.66	0.69	0.46
Rotten Tomatoes	MSE (RMSE)	0.07 (0.26)	**0.07 (0.26)**	**0.15 (0.39)**
	MAE	0.19	0.19	0.31
	R^2	0.50	0.52	-0.10
Amazon	MSE (RMSE)	0.05 (0.23)	**0.04 (0.21)**	**0.06 (0.25)**
	MAE	0.17	0.15	0.18
	R^2	0.63	0.70	0.57

Table 14. *Evaluation metrics for SVR (testing set - Spotify). Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Training Set	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2Vec)	LR (Word2Vec)	RNN
Spotify	Accuracy	0.8776	0.8535	0.8779	0.8814	0.8581	0.8804	0.8633	0.8612	0.88
	F1 score	0.89; 0.86	0.87; 0.84	0.89; 0.86	0.89; 0.87	0.88; 0.84	0.89; 0.86	0.88; 0.84	0.88; 0.84	0.86
	Precision	0.90; 0.95	0.89; 0.81	0.90; 0.85	0.89; 0.87	0.86; 0.85	0.89; 0.87	0.87; 0.86	0.87; 0.85	0.88
	Recall	0.88; 0.88	0.84; 0.87	0.88; 0.88	0.90; 0.86	0.89; 0.82	0.90; 0.86	0.89; 0.83	0.89; 0.82	0.85
IMDB	Accuracy	0.703	**0.652**	0.708	0.704	**0.643**	0.7	0.649	0.65	0.71
	F1 score	0.69; 0.71	0.61; 0.68	0.70; 0.71	0.70; 0.71	0.60; 0.68	0.69; 0.71	0.59; 0.70	0.59; 0.70	0.69
	Precision	0.82; 0.62	0.81; 0.57	0.81; 0.63	0.82; 0.62	0.80; 0.56	0.83; 0.62	0.86; 0.56	0.86; 0.56	0.66
	Recall	0.60; 0.83	0.50; 0.85	0.62; 0.82	0.60; 0.83	0.48; 0.85	0.59; 0.84	0.44; 0.91	0.45; 0.91	0.71
Twitter	Accuracy	**0.775**	**0.794**	**0.782**	**0.768**	**0.783**	**0.787**	**0.774**	**0.787**	0.72
	F1 score	0.80; 0.73	0.82; 0.76	0.79; 0.77	0.80; 0.71	0.82; 0.74	0.80; 0.77	0.81; 0.72	0.81; 0.75	0.67
	Precision	0.78; 0.77	0.80; 0.78	0.84; 0.72	0.76; 0.78	0.78; 0.79	0.82; 0.74	0.76; 0.79	0.80; 0.77	0.73
	Recall	0.83; 0.71	0.83; 0.74	0.75; 0.83	0.86; 0.65	0.86; 0.69	0.79; 0.79	0.86; 0.66	0.82; 0.74	0.7
Rotten Tomatoes	Accuracy	**0.674**	0.701	**0.679**	**0.687**	0.704	**0.691**	**0.601**	**0.486**	**0.6**
	F1 score	0.68; 0.67	0.72; 0.68	0.68; 0.68	0.69; 0.68	0.72; 0.68	0.70; 0.69	0.62; 0.57	0.53; 0.44	0.66
	Precision	0.76; 0.61	0.76; 0.64	0.77; 0.61	0.77; 0.62	0.76; 0.65	0.77; 0.62	0.66; 0.54	0.54; 0.42	0.53
	Recall	0.61; 0.75	0.68; 0.73	0.61; 0.77	0.63; 0.76	0.69; 0.72	0.63; 0.76	0.60; 0.61	0.51; 0.45	0.87
Amazon	Accuracy	0.705	0.701	0.718	0.731	0.701	0.733	0.753	0.754	**0.76**
	F1 score	0.67; 0.74	0.67; 0.73	0.69; 0.74	0.72; 0.74	0.67; 0.73	0.72; 0.75	0.75; 0.76	0.75; 0.76	0.76
	Precision	0.91; 0.61	0.89; 0.61	0.90; 0.62	0.87; 0.64	0.88; 0.61	0.87; 0.64	0.87; 0.67	0.87; 0.67	0.67
	Recall	0.53; 0.93	0.53; 0.91	0.56; 0.92	0.61; 0.89	0.54; 0.91	0.61; 0.89	0.66; 0.87	0.66; 0.87	0.88

Table 15. *Evaluation metrics for machine learning models (testing set - Spotify). Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Training set	Evaluation	SVR (BOW)	SVR (TF-IDF)	SVR (Word2Vec)
Amazon	MSE (RMSE)	0.02 (0.14)	0.02 (0.14)	0.03 (0.17)
	MAE	0.10	0.10	0.12
	R^2	0.68	0.69	0.53
IMDB	MSE (RMSE)	**0.03 (0.17)**	**0.02 (0.16)**	**0.03 (0.17)**
	MAE	0.11	0.11	0.13
	R^2	0.55	0.59	0.51
Twitter	MSE (RMSE)	0.04 (0.19)	**0.04 (0.19)**	0.04 (0.21)
	MAE	0.14	0.14	0.15
	R^2	0.41	0.42	0.31
Rotten Tomatoes	MSE (RMSE)	**0.04 (0.20)**	0.03 (0.17)	**0.06 (0.25)**
	MAE	0.15	0.13	0.19
	R^2	0.32	0.51	-0.05
Spotify	MSE (RMSE)	0.03 (0.18)	0.03 (0.18)	0.04 (0.19)
	MAE	0.13	0.14	0.14
	R^2	0.44	0.46	0.42

Table 16. *Evaluation metrics for SVR (testing set - Amazon). Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Training Set	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2Vec)	LR (Word2Vec)	RNN
Amazon	Accuracy	0.825	0.807	0.826	0.828	0.810	0.826	0.808	0.805	0.49
	F1 score	0.82; 0.83	0.80; 0.81	0.82; 0.83	0.82; 0.83	0.80; 0.82	0.82; 0.83	0.80; 0.81	0.80; 0.81	0.65
	Precision	0.84; 0.82	0.80; 0.81	0.83; 0.82	0.83; 0.82	0.82; 0.80	0.83; 0.82	0.80; 0.81	0.80; 0.81	0.49
	Recall	0.80; 0.85	0.80; 0.82	0.80; 0.85	0.81; 0.85	0.79; 0.83	0.81; 0.84	0.80; 0.81	0.80; 0.81	1
IMDB	Accuracy	**0.767**	**0.760**	**0.768**	**0.770**	**0.760**	**0.773**	**0.784**	**0.784**	**0.77**
	F1 score	0.75; 0.78	0.75; 0.77	0.75; 0.78	0.76; 0.78	0.75; 0.77	0.76; 0.78	0.76; 0.80	0.76; 0.80	0.78
	Precision	0.78; 0.75	0.77; 0.75	0.78; 0.76	0.77; 0.77	0.77; 0.75	0.78; 0.77	0.82; 0.76	0.82; 0.76	0.77
	Recall	0.72; 0.81	0.72; 0.79	0.73; 0.80	0.74; 0.79	0.72; 0.80	0.75; 0.80	0.72; 0.85	0.71; 0.85	0.78
Twitter	Accuracy	**0.678**	**0.677**	**0.684**	**0.682**	**0.680**	**0.685**	0.652	0.644	**0.62**
	F1 score	0.66; 0.69	0.62; 0.72	0.64; 0.72	0.67; 0.69	0.64; 0.71	0.64; 0.72	0.65; 0.65	0.60; 0.68	0.69
	Precision	0.67; 0.68	0.73; 0.65	0.71; 0.67	0.67; 0.69	0.71; 0.66	0.72; 0.66	0.64; 0.67	0.66; 0.63	0.6
	Recall	0.65; 0.70	0.54; 0.81	0.59; 0.78	0.68; 0.69	0.59; 0.77	0.58; 0.78	0.66; 0.64	0.54; 0.74	0.82
Rotten Tomatoes	Accuracy	0.691	0.689	0.698	0.696	0.684	0.702	**0.563**	**0.489**	0.68
	F1 score	0.69; 0.69	0.72; 0.64	0.70; 0.69	0.70; 0.69	0.72; 0.63	0.72; 0.68	0.61; 0.50	0.48; 0.50	0.71
	Precision	0.67; 0.71	0.64; 0.78	0.68; 0.72	0.67; 0.72	0.63; 0.79	0.66; 0.75	0.54; 0.60	0.47; 0.50	0.66
	Recall	0.71; 0.67	0.83; 0.55	0.73; 0.67	0.73; 0.66	0.85; 0.52	0.78; 0.62	0.70; 0.43	0.48; 0.49	0.76
Spotify	Accuracy	0.737	0.726	0.737	0.740	0.717	0.740	0.708	0.705	0.74
	F1 score	0.72; 0.75	0.72; 0.74	0.72; 0.75	0.74; 0.74	0.74; 0.69	0.74; 0.74	0.70; 0.71	0.71; 0.71	0.72
	Precision	0.74; 0.73	0.72; 0.73	0.74; 0.73	0.72; 0.76	0.67; 0.78	0.72; 0.76	0.69; 0.72	0.69; 0.72	0.8
	Recall	0.71; 0.77	0.71; 0.74	0.70; 0.77	0.76; 0.72	0.82; 0.62	0.77; 0.72	0.72; 0.70	0.72; 0.69	0.65

Table 17. *Evaluation metrics for machine learning models (testing set - Amazon). Best performing cross-domain models in* ***green****, worst - in* ***red****.*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Rotten Tomatoes	Not Removed	Accuracy	0.75834	0.7123	0.76236	0.76524	0.7089	0.75652	0.60042	0.52484	0.76
		F1 score	0.78; 0.73	0.77; 0.63	0.78; 0.74	0.78; 0.75	0.76; 0.62	0.78; 0.72	0.67; 0.49	0.63; 0.35	0.75
		Precision	0.72; 0.82	0.65; 0.89	0.72; 0.83	0.73; 0.81	0.64; 0.90	0.70; 0.85	0.57; 0.68	0.52; 0.55	0.77
		Recall	0.85; 0.66	0.94; 0.48	0.86; 0.66	0.83; 0.70	0.94; 0.47	0.89; 0.63	0.82; 0.38	0.80; 0.25	0.74
	Training set	Accuracy	0.76322	0.72354	0.76818	0.76874	0.71806	0.76072	0.5563	0.52336	0.68
		F1 score	0.78; 0.74	0.77; 0.65	0.79; 0.74	0.79; 0.75	0.77; 0.64	0.79; 0.72	0.66; 0.35	0.64; 0.31	0.67
		Precision	0.72; 0.82	0.66; 0.89	0.72; 0.83	0.73; 0.82	0.65; 0.89	0.71; 0.86	0.53; 0.65	0.51; 0.56	0.69
		Recall	0.86; 0.67	0.94; 0.51	0.87; 0.67	0.85; 0.69	0.94; 0.50	0.89; 0.63	0.87; 0.24	0.84; 0.21	0.65
	Training + Testing sets	Accuracy	0.76322	0.72354	0.76818	0.76874	0.71806	0.76072	0.5563	0.52336	0.68
		F1 score	0.78; 0.74	0.77; 0.65	0.79; 0.74	0.79; 0.75	0.77; 0.64	0.79; 0.72	0.66; 0.35	0.64; 0.31	0.67
		Precision	0.72; 0.82	0.66; 0.89	0.72; 0.83	0.73; 0.82	0.65; 0.89	0.71; 0.86	0.53; 0.65	0.51; 0.56	0.69
		Recall	0.86; 0.67	0.94; 0.51	0.87; 0.67	0.85; 0.69	0.94; 0.50	0.89; 0.63	0.87; 0.24	0.84; 0.21	0.65

Table 18. *Evaluation metrics for ML models with domain-specific words removed (training set - Rotten Tomatoes, testing set - IMDB)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Spotify	Not Removed	Accuracy	0.73844	0.72036	0.74048	0.7267	0.68666	0.72306	0.70314	0.69796	0.73
		F1 score	0.75; 0.72	0.74; 0.69	0.75; 0.73	0.76; 0.68	0.74; 0.60	0.76; 0.68	0.73; 0.67	0.72; 0.67	0.71
		Precision	0.72; 0.77	0.69; 0.77	0.72; 0.77	0.68; 0.81	0.63; 0.83	0.68; 0.81	0.67; 0.75	0.67; 0.75	0.77
		Recall	0.79; 0.68	0.81; 0.63	0.79; 0.69	0.86; 0.59	0.90; 0.47	0.86; 0.59	0.79; 0.61	0.80; 0.60	0.66
	Training set	Accuracy	0.73194	0.7222	0.73392	0.7139	0.70282	0.71272	0.6844	0.67596	0.72
		F1 score	0.76; 0.70	0.72; 0.73	0.76; 0.70	0.76; 0.65	0.75; 0.64	0.76; 0.65	0.73; 0.62	0.73; 0.60	0.66
		Precision	0.69; 0.79	0.73; 0.71	0.69; 0.80	0.66; 0.83	0.65; 0.80	0.66; 0.83	0.64; 0.78	0.63; 0.79	0.82
		Recall	0.84; 0.63	0.70; 0.74	0.84; 0.63	0.89; 0.54	0.87; 0.54	0.89; 0.54	0.85; 0.52	0.87; 0.48	0.56
	Training + Testing sets	Accuracy	0.73302	0.72256	0.73558	0.71616	0.70578	0.71582	0.67614	0.67082	0.69
		F1 score	0.76; 0.70	0.71; 0.73	0.76; 0.71	0.76; 0.66	0.75; 0.65	0.76; 0.66	0.73; 0.59	0.73; 0.58	0.60
		Precision	0.69; 0.79	0.74; 0.71	0.70; 0.80	0.66; 0.83	0.66; 0.80	0.66; 0.83	0.63; 0.80	0.62; 0.80	0.85
		Recall	0.84; 0.63	0.70; 0.75	0.84; 0.63	0.89; 0.54	0.87; 0.55	0.89; 0.54	0.88; 0.47	0.88; 0.46	0.46

Table 19. *Evaluation metrics for for ML models with domain specific words removed (training set - Spotify, testing set - IMDB)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
IMDB	Not Removed	Accuracy	0.621	0.628	0.625	0.626	0.624	0.629	0.67	0.67	0.62
		F1 score	0.64; 0.61	0.65; 0.60	0.65; 0.60	0.64; 0.61	0.65; 0.60	0.64; 0.61	0.67; 0.67	0.67; 0.67	0.68
		Precision	0.64; 0.61	0.63; 0.62	0.63; 0.62	0.64; 0.61	0.63; 0.62	0.64; 0.61	0.69; 0.65	0.69; 0.65	0.57
		Recall	0.63; 0.61	0.68; 0.57	0.66; 0.58	0.64; 0.61	0.67; 0.58	0.65; 0.61	0.65; 0.69	0.65; 0.69	0.83
	Training set	Accuracy	0.63	0.635	0.633	0.633	0.632	0.636	0.671	0.674	0.62
		F1 score	0.64; 0.62	0.66; 0.60	0.65; 0.61	0.65; 0.62	0.66; 0.61	0.65; 0.62	0.67; 0.67	0.67; 0.67	0.55
		Precision	0.64; 0.61	0.64; 0.63	0.64; 0.62	0.65; 0.62	0.64; 0.62	0.65; 0.62	0.70; 0.65	0.70; 0.65	0.64
		Recall	0.64; 0.62	0.69; 0.58	0.67; 0.59	0.65; 0.61	0.67; 0.59	0.66; 0.62	0.65; 0.69	0.65; 0.70	0.49
	Training + Testing sets	Accuracy	0.629	0.634	0.632	0.632	0.631	0.635	0.668	0.669	0.61
		F1 score	0.64; 0.62	0.66; 0.61	0.65; 0.61	0.65; 0.62	0.65; 0.61	0.65; 0.62	0.67; 0.67	0.67; 0.67	0.67
		Precision	0.64; 0.61	0.64; 0.63	0.64; 0.62	0.65; 0.62	0.64; 0.62	0.65; 0.62	0.69; 0.64	0.70; 0.64	0.57
		Recall	0.64; 0.62	0.68; 0.59	0.67; 0.59	0.65; 0.61	0.67; 0.59	0.65; 0.61	0.64; 0.69	0.64; 0.70	0.83

Table 20. *Evaluation metrics for ML models with domain-specific words removed (training set - IMDB, testing set - Twitter)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Rotten Tomatoes	Not Removed	Accuracy	0.591	0.598	0.594	0.597	0.6	0.603	0.526	0.453	0.57
		F1 score	0.66; 0.48	0.67; 0.49	0.66; 0.49	0.66; 0.49	0.67; 0.49	0.67; 0.51	0.55; 0.50	0.36; 0.52	0.63
		Precision	0.58; 0.62	0.58; 0.63	0.58; 0.62	0.59; 0.62	0.59; 0.64	0.59; 0.63	0.54; 0.51	0.46; 0.45	0.54
		Recall	0.77; 0.40	0.78; 0.40	0.76; 0.41	0.77; 0.41	0.79; 0.39	0.76; 0.43	0.55; 0.50	0.30; 0.62	0.76
	Training set	Accuracy	0.594	0.599	0.598	0.601	0.602	0.608	0.488	0.461	0.58
		F1 score	0.66; 0.49	0.67; 0.49	0.66; 0.50	0.67; 0.50	0.67; 0.49	0.67; 0.51	0.51; 0.47	0.47; 0.46	0.54
		Precision	0.58; 0.62	0.59; 0.63	0.59; 0.62	0.59; 0.63	0.59; 0.64	0.59; 0.63	0.51; 0.47	0.48; 0.44	0.57
		Recall	0.77; 0.41	0.78; 0.40	0.76; 0.42	0.77; 0.42	0.79; 0.39	0.77; 0.43	0.51; 0.47	0.45; 0.47	0.51
	Training + Testing sets	Accuracy	0.594	0.6	0.598	0.6	0.602	0.607	0.481	0.457	
		F1 score	0.66; 0.49	0.67; 0.49	0.66; 0.50	0.67; 0.50	0.67; 0.49	0.67; 0.52	0.49; 0.47	0.45; 0.46	
		Precision	0.58; 0.62	0.59; 0.63	0.59; 0.62	0.59; 0.63	0.59; 0.64	0.60; 0.63	0.50; 0.46	0.47; 0.44	
		Recall	0.76; 0.41	0.78; 0.41	0.76; 0.42	0.77; 0.42	0.79; 0.40	0.76; 0.44	0.48; 0.48	0.43; 0.49	

Table 21. *Evaluation metrics for ML models with domain-specific words removed (training set - Rotten Tomatoes, testing set - Twitter)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Spotify	Not Removed	Accuracy	0.689	0.6902	0.688	0.689	0.679	0.693	0.67	0.669	0.7
		F1 score	0.68; 0.70	0.74; 0.62	0.67; 0.70	0.69; 0.69	0.74; 0.58	0.70; 0.69	0.70; 0.64	0.70; 0.63	0.68
		Precision	0.74; 0.65	0.66; 0.76	0.74; 0.65	0.72; 0.66	0.64; 0.78	0.71; 0.67	0.67; 0.68	0.66; 0.68	0.7
		Recall	0.62; 0.76	0.85; 0.52	0.62; 0.76	0.66; 0.73	0.88; 0.46	0.69; 0.70	0.74; 0.60	0.74; 0.59	0.66
	Training set	Accuracy	0.691	0.691	0.689	0.688	0.679	0.691	0.682	0.676	0.7
		F1 score	0.69; 0.69	0.73; 0.63	0.68; 0.69	0.69; 0.68	0.74; 0.59	0.70; 0.68	0.72; 0.64	0.71; 0.63	0.68
		Precision	0.73; 0.66	0.67; 0.74	0.72; 0.66	0.71; 0.67	0.64; 0.77	0.70; 0.68	0.67; 0.70	0.66; 0.70	0.71
		Recall	0.65; 0.73	0.82; 0.56	0.65; 0.73	0.68; 0.70	0.87; 0.48	0.70; 0.68	0.77; 0.58	0.77; 0.57	0.65
	Training + Testing sets	Accuracy	0.689	0.691	0.688	0.685	0.68	0.689	0.673	0.671	0.7
		F1 score	0.68; 0.69	0.73; 0.63	0.68; 0.69	0.69; 0.68	0.74; 0.59	0.70; 0.68	0.71; 0.63	0.71; 0.62	0.67
		Precision	0.73; 0.66	0.66; 0.74	0.72; 0.66	0.71; 0.66	0.64; 0.77	0.70; 0.67	0.66; 0.69	0.65; 0.70	0.72
		Recall	0.65; 0.74	0.82; 0.55	0.65; 0.73	0.67; 0.70	0.87; 0.48	0.70; 0.68	0.77; 0.57	0.78; 0.55	0.63

Table 22. *Evaluation metrics for ML models with domain-specific words removed (training set - Spotify, testing set - Twitter)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Amazon	Not Removed	Accuracy	0.638	0.666	0.646	0.679	0.662	0.681	0.702	0.705	0.68
		F1 score	0.57; 0.69	0.66; 0.68	0.60; 0.68	0.69; 0.66	0.64; 0.68	0.70; 0.66	0.72; 0.68	0.73; 0.68	0.69
		Precision	0.74; 0.59	0.71; 0.63	0.73; 0.60	0.69; 0.67	0.71; 0.62	0.69; 0.67	0.70; 0.70	0.70; 0.71	0.65
		Recall	0.47; 0.83	0.61; 0.73	0.50; 0.80	0.70; 0.66	0.59; 0.74	0.70; 0.66	0.75; 0.65	0.75; 0.65	0.73
	Training set	Accuracy	0.637	0.66	0.644	0.679	0.657	0.683	0.7	0.704	0.68
		F1 score	0.58; 0.68	0.65; 0.67	0.60; 0.68	0.70; 0.66	0.64; 0.67	0.70; 0.66	0.72; 0.67	0.73; 0.68	0.66
		Precision	0.73; 0.59	0.70; 0.63	0.72; 0.60	0.69; 0.67	0.71; 0.62	0.69; 0.68	0.69; 0.71	0.70; 0.71	0.68
		Recall	0.47; 0.81	0.60; 0.73	0.51; 0.78	0.71; 0.65	0.58; 0.74	0.72; 0.64	0.76; 0.64	0.76; 0.64	0.64
	Training + Testing sets	Accuracy	0.635	0.659	0.642	0.678	0.656	0.682	0.704	0.706	0.68
		F1 score	0.57; 0.68	0.64; 0.67	0.60; 0.68	0.70; 0.66	0.63; 0.68	0.70; 0.66	0.72; 0.68	0.73; 0.68	0.65
		Precision	0.73; 0.59	0.70; 0.62	0.72; 0.60	0.68; 0.67	0.71; 0.62	0.68; 0.68	0.70; 0.71	0.70; 0.71	0.69
		Recall	0.47; 0.81	0.59; 0.73	0.51; 0.78	0.71; 0.65	0.57; 0.74	0.72; 0.64	0.75; 0.65	0.75; 0.65	0.61

Table 23. *Evaluation metrics for ML models with domain-specific words removed (training set - Amazon, testing set - Twitter)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
IMDB	Not Removed	Accuracy	0.645	0.655	0.647	0.650	0.657	0.651	0.717	0.717	0.67
		F1 score	0.64; 0.65	0.63; 0.68	0.65; 0.64	0.65; 0.65	0.63; 0.68	0.65; 0.65	0.70; 0.73	0.70; 0.73	0.70
		Precision	0.65; 0.64	0.68; 0.64	0.64; 0.65	0.65; 0.65	0.68; 0.64	0.65; 0.65	0.75; 0.69	0.75; 0.69	0.65
		Recall	0.63; 0.66	0.59; 0.72	0.67; 0.63	0.65; 0.65	0.59; 0.72	0.64; 0.66	0.66; 0.77	0.66; 0.77	0.75
	Training set	Accuracy	0.645	0.652	0.646	0.650	0.657	0.652	0.716	0.717	0.69
		F1 score	0.64; 0.65	0.63; 0.67	0.65; 0.64	0.65; 0.65	0.64; 0.67	0.65; 0.65	0.70; 0.73	0.70; 0.73	0.72
		Precision	0.65; 0.64	0.67; 0.64	0.64; 0.65	0.65; 0.65	0.67; 0.64	0.65; 0.65	0.75; 0.69	0.75; 0.69	0.66
		Recall	0.62; 0.67	0.60; 0.70	0.65; 0.64	0.66; 0.65	0.60; 0.71	0.66; 0.65	0.66; 0.78	0.66; 0.78	0.79
	Training + Testing sets	Accuracy	0.645	0.652	0.646	0.650	0.657	0.652	0.716	0.715	0.69
		F1 score	0.64; 0.65	0.63; 0.67	0.65; 0.64	0.65; 0.65	0.64; 0.67	0.65; 0.65	0.70; 0.73	0.70; 0.73	0.71
		Precision	0.65; 0.64	0.67; 0.64	0.64; 0.65	0.65; 0.65	0.67; 0.64	0.65; 0.65	0.74; 0.69	0.74; 0.69	0.67
		Recall	0.62; 0.67	0.60; 0.70	0.65; 0.64	0.66; 0.65	0.60; 0.71	0.66; 0.65	0.66; 0.77	0.66; 0.77	0.77

Table 24. *Evaluation metrics for ML models with domain-specific words removed (training set - IMDB, testing set - Rotten Tomatoes)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Twitter	Not Removed	Accuracy	0.548	0.553	0.559	0.552	0.553	0.562	0.536	0.549	0.57
		F1 score	0.62; 0.44	0.54; 0.57	0.50; 0.60	0.58; 0.51	0.55; 0.56	0.51; 0.61	0.58; 0.49	0.54; 0.56	0.61
		Precision	0.53; 0.58	0.56; 0.55	0.58; 0.55	0.54; 0.56	0.55; 0.55	0.58; 0.55	0.53; 0.55	0.55; 0.55	0.55
		Recall	0.74; 0.35	0.52; 0.58	0.45; 0.67	0.63; 0.47	0.54; 0.56	0.45; 0.68	0.64; 0.44	0.53; 0.56	0.68
	Training set	Accuracy	0.547	0.56	0.548	0.551	0.562	0.55	0.542	0.547	0.58
		F1 score	0.63; 0.42	0.54; 0.58	0.56; 0.53	0.60; 0.49	0.55; 0.57	0.56; 0.54	0.60; 0.47	0.58; 0.51	0.6
		Precision	0.53; 0.58	0.57; 0.56	0.54; 0.55	0.54; 0.57	0.57; 0.56	0.55; 0.55	0.53; 0.56	0.54; 0.56	0.58
		Recall	0.77; 0.32	0.51; 0.61	0.59; 0.51	0.67; 0.43	0.53; 0.59	0.57; 0.52	0.68; 0.41	0.62; 0.47	0.62
	Training + Testing sets	Accuracy	0.55	0.564	0.554	0.558	0.566	0.556	0.539	0.545	0.58
		F1 score	0.64; 0.39	0.59; 0.54	0.59; 0.52	0.62; 0.47	0.60; 0.53	0.59; 0.52	0.60; 0.45	0.59; 0.49	0.61
		Precision	0.53; 0.60	0.56; 0.57	0.55; 0.56	0.54; 0.59	0.56; 0.58	0.55; 0.57	0.53; 0.56	0.54; 0.56	0.57
		Recall	0.81; 0.29	0.62; 0.51	0.63; 0.48	0.73; 0.39	0.64; 0.49	0.64; 0.48	0.70; 0.38	0.65; 0.44	0.65

Table 25. *Evaluation metrics for ML models with domain-specific words removed (training set - Twitter, testing set - Rotten Tomatoes)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Spotify	Not Removed	Accuracy	0.597	0.583	0.597	0.605	0.584	0.604	0.588	0.587	0.64
		F1 score	0.53; 0.64	0.64; 0.51	0.53; 0.65	0.57; 0.64	0.66; 0.47	0.59; 0.62	0.60; 0.58	0.60; 0.57	0.62
		Precision	0.63; 0.58	0.56; 0.62	0.63; 0.58	0.63; 0.59	0.56; 0.65	0.61; 0.60	0.58; 0.59	0.58; 0.59	0.65
		Recall	0.46; 0.73	0.76; 0.43	0.46; 0.73	0.52; 0.69	0.80; 0.37	0.56; 0.65	0.61; 0.57	0.62; 0.55	0.59
	Training set	Accuracy	0.602	0.582	0.603	0.606	0.584	0.607	0.586	0.586	0.64
		F1 score	0.56; 0.63	0.62; 0.53	0.56; 0.64	0.59; 0.63	0.65; 0.49	0.60; 0.61	0.61; 0.56	0.61; 0.56	0.60
		Precision	0.62; 0.59	0.59; 0.60	0.63; 0.59	0.62; 0.60	0.56; 0.63	0.61; 0.60	0.58; 0.60	0.58; 0.60	0.66
		Recall	0.51; 0.69	0.69; 0.48	0.51; 0.69	0.55; 0.66	0.77; 0.40	0.59; 0.62	0.64; 0.53	0.65; 0.52	0.55
	Training + Testing sets	Accuracy	0.602	0.582	0.602	0.606	0.583	0.608	0.587	0.59	0.63
		F1 score	0.56; 0.63	0.62; 0.53	0.56; 0.64	0.58; 0.63	0.65; 0.49	0.60; 0.62	0.62; 0.55	0.62; 0.55	0.59
		Precision	0.62; 0.59	0.57; 0.60	0.63; 0.59	0.62; 0.60	0.56; 0.63	0.61; 0.60	0.57; 0.60	0.58; 0.61	0.67
		Recall	0.51; 0.69	0.69; 0.48	0.51; 0.69	0.55; 0.66	0.77; 0.40	0.59; 0.63	0.67; 0.51	0.67; 0.51	0.54

Table 26. *Evaluation metrics for ML models with domain-specific words removed (training set - Spotify, testing set - Rotten Tomatoes)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Amazon	Not Removed	Accuracy	0.628	0.64	0.63	0.632	0.638	0.636	0.676	0.676	0.69
		F1 score	0.56; 0.68	0.61; 0.67	0.58; 0.67	0.64; 0.63	0.60; 0.67	0.64; 0.63	0.67; 0.68	0.67; 0.68	0.70
		Precision	0.69; 0.60	0.66; 0.62	0.67; 0.60	0.63; 0.63	0.67; 0.62	0.63; 0.64	0.68; 0.67	0.68; 0.67	0.68
		Recall	0.47; 0.79	0.56; 0.72	0.51; 0.75	0.64; 0.62	0.54; 0.74	0.65; 0.62	0.67; 0.69	0.67; 0.68	0.72
	Training set	Accuracy	0.632	0.642	0.638	0.637	0.637	0.637	0.678	0.675	0.69
		F1 score	0.57; 0.68	0.61; 0.67	0.59; 0.67	0.64; 0.63	0.60; 0.67	0.64; 0.63	0.67; 0.68	0.67; 0.68	0.68
		Precision	0.69; 0.60	0.67; 0.62	0.68; 0.61	0.63; 0.64	0.67; 0.61	0.63; 0.64	0.68; 0.67	0.68; 0.67	0.70
		Recall	0.49; 0.78	0.57; 0.72	0.53; 0.74	0.65; 0.62	0.54; 0.74	0.66; 0.62	0.67; 0.69	0.67; 0.68	0.67
	Training + Testing sets	Accuracy	0.629	0.642	0.633	0.635	0.638	0.636	0.678	0.681	0.68
		F1 score	0.56; 0.68	0.60; 0.68	0.58; 0.67	0.64; 0.63	0.59; 0.68	0.64; 0.63	0.68; 0.68	0.68; 0.68	0.69
		Precision	0.69; 0.60	0.68; 0.62	0.68; 0.61	0.63; 0.64	0.68; 0.61	0.63; 0.64	0.68; 0.68	0.68; 0.68	0.68
		Recall	0.46; 0.80	0.53; 0.75	0.51; 0.76	0.64; 0.63	0.51; 0.76	0.65; 0.62	0.67; 0.68	0.67; 0.69	0.71

Table 27. *Evaluation metrics for ML models with domain-specific words removed (training set - Amazon, testing set - Rotten Tomatoes)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
IMDB	Not Removed	Accuracy	0.703	0.652	0.708	0.704	0.643	0.700	0.649	0.650	0.710
		F1 score	0.69; 0.71	0.61; 0.68	0.70; 0.71	0.70; 0.71	0.60; 0.68	0.69; 0.71	0.59; 0.70	0.59; 0.70	0.69
		Precision	0.82; 0.62	0.81; 0.57	0.81; 0.63	0.82; 0.62	0.80; 0.56	0.83; 0.62	0.86; 0.56	0.86; 0.56	0.66
		Recall	0.60; 0.83	0.50; 0.85	0.62; 0.82	0.60; 0.83	0.48; 0.85	0.59; 0.84	0.44; 0.91	0.45; 0.91	0.71
	Training set	Accuracy	0.711	0.663	0.713	0.713	0.651	0.709	0.685	0.685	0.700
		F1 score	0.71; 0.72	0.64; 0.68	0.71; 0.71	0.71; 0.72	0.62; 0.68	0.70; 0.72	0.65; 0.71	0.65; 0.71	0.71
		Precision	0.82; 0.63	0.79; 0.58	0.81; 0.64	0.81; 0.64	0.79; 0.57	0.82; 0.63	0.85; 0.60	0.85; 0.60	0.61
		Recall	0.62; 0.82	0.54; 0.82	0.64; 0.81	0.63; 0.82	0.52; 0.82	0.61; 0.83	0.53; 0.88	0.53; 0.88	0.86
	Training + Testing sets	Accuracy	0.714	0.698	0.720	0.718	0.693	0.718	0.731	0.730	0.710
		F1 score	0.71; 0.72	0.70; 0.70	0.72; 0.72	0.72; 0.72	0.69; 0.69	0.71; 0.72	0.72; 0.74	0.72; 0.74	0.74
		Precision	0.82; 0.64	0.79; 0.62	0.82; 0.65	0.82; 0.64	0.79; 0.62	0.82; 0.64	0.85; 0.65	0.85; 0.65	0.62
		Recall	0.62; 0.83	0.62; 0.79	0.65; 0.81	0.64; 0.82	0.62; 0.79	0.63; 0.83	0.63; 0.86	0.63; 0.86	0.90

Table 28. *Evaluation metrics for ML models with domain-specific words removed (training set - IMDB, testing set - Spotify)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Twitter	Not Removed	Accuracy	0.775	0.794	0.782	0.768	0.783	0.787	0.774	0.787	0.72
		F1 score	0.80; 0.73	0.82; 0.76	0.79; 0.77	0.80; 0.71	0.82; 0.74	0.80; 0.77	0.81; 0.72	0.81; 0.75	0.67
		Precision	0.78; 0.77	0.80; 0.78	0.84; 0.72	0.76; 0.78	0.78; 0.79	0.82; 0.74	0.76; 0.79	0.80; 0.77	0.73
		Recall	0.83; 0.71	0.83; 0.74	0.75; 0.83	0.86; 0.65	0.86; 0.69	0.79; 0.79	0.86; 0.66	0.82; 0.74	0.70
	Training set	Accuracy	0.772	0.794	0.777	0.768	0.783	0.783	0.770	0.776	0.72
		F1 score	0.80; 0.83	0.82; 0.76	0.79; 0.76	0.81; 0.71	0.82; 0.73	0.81; 0.75	0.81; 0.71	0.81; 0.73	0.71
		Precision	0.77; 0.77	0.80; 0.79	0.82; 0.73	0.76; 0.79	0.77; 0.80	0.81; 0.75	0.75; 0.80	0.78; 0.77	0.65
		Recall	0.84; 0.68	0.85; 0.73	0.77; 0.78	0.86; 0.65	0.87; 0.68	0.80; 0.76	0.87; 0.64	0.84; 0.69	0.78
	Training + Testing sets	Accuracy	0.775	0.805	0.760	0.786	0.801	0.782	0.783	0.775	0.74
		F1 score	0.80; 0.74	0.82; 0.78	0.77; 0.75	0.81; 0.75	0.83; 0.77	0.79; 0.77	0.81; 0.75	0.80; 0.75	0.73
		Precision	0.79; 0.75	0.83; 0.77	0.84; 0.69	0.80; 0.77	0.81; 0.79	0.84; 0.72	0.79; 0.77	0.80; 0.74	0.67
		Recall	0.81; 0.74	0.82; 0.79	0.71; 0.83	0.82; 0.74	0.84; 0.75	0.75; 0.82	0.83; 0.72	0.79; 0.76	0.79

Table 29. *Evaluation metrics for ML models with domain-specific words removed (training set - Twitter, testing set - Spotify)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Rotten Tomatoes	Not Removed	Accuracy	0.674	0.701	0.679	0.687	0.704	0.691	0.601	0.486	0.6
		F1 score	0.68; 0.67	0.72; 0.68	0.68; 0.68	0.69; 0.68	0.72; 0.68	0.70; 0.69	0.62; 0.57	0.53; 0.44	0.66
		Precision	0.76; 0.61	0.76; 0.64	0.77; 0.61	0.77; 0.62	0.76; 0.65	0.77; 0.62	0.66; 0.54	0.54; 0.42	0.53
		Recall	0.61; 0.75	0.68; 0.73	0.61; 0.77	0.63; 0.76	0.69; 0.72	0.63; 0.76	0.60; 0.61	0.51; 0.45	0.87
	Training set	Accuracy	0.691	0.702	0.691	0.701	0.704	0.705	0.475	0.472	0.63
		F1 score	0.72; 0.66	0.74; 0.66	0.71; 0.66	0.73; 0.67	0.74; 0.66	0.73; 0.67	0.58; 0.30	0.58; 0.29	0.63
		Precision	0.73; 0.64	0.73; 0.67	0.74; 0.64	0.74; 0.66	0.73; 0.67	0.75; 0.66	0.52; 0.37	0.52; 0.36	0.56
		Recall	0.70; 0.68	0.74; 0.65	0.69; 0.69	0.72; 0.68	0.76; 0.64	0.72; 0.69	0.65; 0.26	0.65; 0.25	0.73
	Training + Testing sets	Accuracy	0.692	0.702	0.692	0.702	0.705	0.705	0.484	0.483	0.68
		F1 score	0.72; 0.66	0.74; 0.66	0.72; 0.66	0.73; 0.67	0.74; 0.65	0.73; 0.67	0.60; 0.28	0.60; 0.28	0.66
		Precision	0.73; 0.64	0.73; 0.67	0.74; 0.64	0.74; 0.66	0.72; 0.68	0.74; 0.66	0.53; 0.37	0.53; 0.36	0.62
		Recall	0.70; 0.68	0.75; 0.65	0.70; 0.69	0.72; 0.68	0.76; 0.63	0.72; 0.69	0.68; 0.23	0.68; 0.23	0.72

Table 30. *Evaluation metrics for ML models with domain-specific words removed (training set - Rotten Tomatoes, testing set - Spotify)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Amazon	Not Removed	Accuracy	0.705	0.701	0.718	0.731	0.701	0.733	0.753	0.754	0.76
		F1 score	0.67; 0.74	0.67; 0.73	0.69; 0.74	0.72; 0.74	0.67; 0.73	0.72; 0.75	0.75; 0.76	0.75; 0.76	0.76
		Precision	0.91; 0.61	0.89; 0.61	0.90; 0.62	0.87; 0.64	0.88; 0.61	0.87; 0.64	0.87; 0.67	0.87; 0.67	0.67
		Recall	0.53; 0.93	0.53; 0.91	0.56; 0.92	0.61; 0.89	0.54; 0.91	0.61; 0.89	0.66; 0.87	0.66; 0.87	0.88
	Training set	Accuracy	0.712	0.698	0.722	0.733	0.699	0.734	0.764	0.766	0.77
		F1 score	0.68; 0.74	0.66; 0.73	0.70; 0.74	0.72; 0.74	0.66; 0.73	0.72; 0.74	0.76; 0.76	0.77; 0.76	0.77
		Precision	0.90; 0.62	0.88; 0.60	0.89; 0.63	0.86; 0.65	0.88; 0.61	0.86; 0.65	0.86; 0.69	0.86; 0.69	0.69
		Recall	0.55; 0.92	0.53; 0.91	0.57; 0.91	0.62; 0.88	0.53; 0.91	0.62; 0.88	0.69; 0.86	0.69; 0.86	0.88
	Training + Testing sets	Accuracy	0.715	0.753	0.727	0.736	0.740	0.739	0.764	0.765	0.77
		F1 score	0.68; 0.74	0.75; 0.76	0.70; 0.75	0.73; 0.74	0.73; 0.75	0.73; 0.74	0.76; 0.76	0.77; 0.76	0.77
		Precision	0.90; 0.62	0.87; 0.67	0.89; 0.63	0.85; 0.65	0.86; 0.65	0.85; 0.66	0.86; 0.69	0.86; 0.69	0.69
		Recall	0.55; 0.92	0.66; 0.87	0.58; 0.91	0.64; 0.86	0.64; 0.87	0.64; 0.86	0.69; 0.86	0.69; 0.86	0.88

Table 31. *Evaluation metrics for ML models with domain-specific words removed (training set - Amazon, testing set - Spotify)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
IMDB	Not Removed	Accuracy	0.767	0.760	0.768	0.770	0.760	0.773	0.784	0.784	0.770
		F1 score	0.75; 0.78	0.75; 0.77	0.75; 0.78	0.76; 0.78	0.75; 0.77	0.76; 0.78	0.76; 0.80	0.76; 0.80	0.78
		Precision	0.78; 0.75	0.77; 0.75	0.78; 0.76	0.77; 0.77	0.77; 0.75	0.78; 0.77	0.82; 0.76	0.82; 0.76	0.77
		Recall	0.72; 0.81	0.72; 0.79	0.73; 0.80	0.74; 0.79	0.72; 0.80	0.75; 0.80	0.72; 0.85	0.71; 0.85	0.78
	Training set	Accuracy	0.769	0.763	0.770	0.772	0.763	0.776	0.786	0.787	0.760
		F1 score	0.75; 0.78	0.75; 0.78	0.76; 0.78	0.76; 0.78	0.75; 0.78	0.77; 0.79	0.77; 0.80	0.77; 0.80	0.75
		Precision	0.78; 0.76	0.78; 0.75	0.78; 0.76	0.77; 0.77	0.78; 0.75	0.78; 0.77	0.81; 0.77	0.81; 0.77	0.79
		Recall	0.73; 0.81	0.72; 0.80	0.74; 0.80	0.75; 0.79	0.72; 0.81	0.75; 0.80	0.74; 0.83	0.74; 0.84	0.71
	Training + Testing sets	Accuracy	0.764	0.762	0.766	0.769	0.761	0.772	0.783	0.784	0.770
		F1 score	0.74; 0.78	0.74; 0.78	0.74; 0.78	0.75; 0.78	0.74; 0.78	0.75; 0.79	0.76; 0.80	0.76; 0.80	0.78
		Precision	0.80; 0.74	0.78; 0.75	0.79; 0.75	0.79; 0.75	0.78; 0.74	0.79; 0.75	0.81; 0.76	0.81; 0.76	0.76
		Recall	0.69; 0.83	0.71; 0.81	0.70; 0.83	0.71; 0.82	0.71; 0.81	0.72; 0.83	0.72; 0.84	0.72; 0.84	0.80

Table 32. *Evaluation metrics for ML models with domain-specific words removed (training set - IMDB, testing set - Amazon)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Twitter	Not Removed	Accuracy	0.678	0.677	0.684	0.682	0.680	0.685	0.652	0.644	0.620
		F1 score	0.66; 0.69	0.62; 0.72	0.64; 0.72	0.67; 0.69	0.64; 0.71	0.64; 0.72	0.65; 0.65	0.60; 0.68	0.69
		Precision	0.67; 0.68	0.73; 0.65	0.71; 0.67	0.67; 0.69	0.71; 0.66	0.72; 0.66	0.64; 0.67	0.66; 0.63	0.60
		Recall	0.65; 0.70	0.54; 0.81	0.59; 0.78	0.68; 0.69	0.59; 0.77	0.58; 0.78	0.66; 0.64	0.54; 0.74	0.82
	Training set	Accuracy	0.676	0.679	0.687	0.687	0.681	0.689	0.654	0.650	0.630
		F1 score	0.64; 0.70	0.61; 0.72	0.64; 0.72	0.67; 0.70	0.63; 0.72	0.65; 0.72	0.67; 0.63	0.63; 0.67	0.68
		Precision	0.69; 0.67	0.74; 0.65	0.72; 0.67	0.69; 0.69	0.72; 0.66	0.72; 0.67	0.62; 0.70	0.65; 0.65	0.61
		Recall	0.60; 0.74	0.53; 0.82	0.58; 0.78	0.65; 0.72	0.57; 0.79	0.58; 0.79	0.73; 0.58	0.62; 0.68	0.78
	Training + Testing sets	Accuracy	0.673	0.677	0.679	0.679	0.678	0.683	0.659	0.650	0.640
		F1 score	0.66; 0.69	0.62; 0.72	0.64; 0.71	0.67; 0.69	0.64; 0.71	0.64; 0.72	0.67; 0.65	0.62; 0.68	0.67
		Precision	0.67; 0.68	0.72; 0.65	0.71; 0.66	0.67; 0.69	0.70; 0.66	0.71; 0.66	0.63; 0.69	0.66; 0.64	0.63
		Recall	0.65; 0.70	0.55; 0.80	0.58; 0.77	0.67; 0.69	0.60; 0.76	0.58; 0.78	0.70; 0.62	0.58; 0.71	0.72

Table 33. *Evaluation metrics for ML models with domain-specific words removed (training set - Twitter, testing set - Amazon)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Rotten Tomatoes	Not Removed	Accuracy	0.691	0.689	0.698	0.696	0.684	0.702	0.563	0.489	0.680
		F1 score	0.69; 0.69	0.72; 0.64	0.70; 0.69	0.70; 0.69	0.72; 0.63	0.72; 0.68	0.61; 0.50	0.48; 0.50	0.71
		Precision	0.67; 0.71	0.64; 0.78	0.68; 0.72	0.67; 0.72	0.63; 0.79	0.66; 0.75	0.54; 0.60	0.47; 0.50	0.66
		Recall	0.71; 0.67	0.83; 0.55	0.73; 0.67	0.73; 0.66	0.85; 0.52	0.78; 0.62	0.70; 0.43	0.48; 0.49	0.76
	Training set	Accuracy	0.700	0.695	0.706	0.705	0.689	0.710	0.504	0.484	0.670
		F1 score	0.70; 0.70	0.73; 0.65	0.71; 0.70	0.71; 0.70	0.73; 0.64	0.72; 0.69	0.59; 0.37	0.57; 0.35	0.69
		Precision	0.68; 0.72	0.64; 0.78	0.68; 0.73	0.68; 0.73	0.63; 0.80	0.67; 0.76	0.49; 0.53	0.48; 0.50	0.66
		Recall	0.72; 0.68	0.84; 0.56	0.73; 0.68	0.74; 0.67	0.86; 0.53	0.78; 0.64	0.73; 0.29	0.71; 0.27	0.73
	Training + Testing sets	Accuracy	0.698	0.697	0.705	0.702	0.694	0.709	0.505	0.486	0.690
		F1 score	0.70; 0.70	0.73; 0.66	0.71; 0.70	0.71; 0.70	0.73; 0.65	0.72; 0.69	0.60; 0.36	0.58; 0.34	0.73
		Precision	0.68; 0.72	0.65; 0.78	0.68; 0.73	0.68; 0.73	0.64; 0.79	0.67; 0.75	0.49; 0.54	0.48; 0.50	0.66
		Recall	0.73; 0.67	0.84; 0.56	0.74; 0.68	0.74; 0.67	0.85; 0.55	0.78; 0.65	0.76; 0.27	0.73; 0.25	0.81

Table 34. *Evaluation metrics for ML models with domain-specific words removed (training set - Rotten Tomatoes, testing set - Amazon)*

Training Set	Domain Specific Words Removed from	Evaluation	SVM (BOW)	NB (BOW)	LR (BOW)	SVM (TF-IDF)	NB (TF-IDF)	LR (TF-IDF)	SVM (Word2vec)	LR (Word2vec)	RNN
Spotify	Not Removed	Accuracy	0.737	0.726	0.737	0.740	0.717	0.740	0.708	0.705	0.740
		F1 score	0.72; 0.75	0.72; 0.74	0.72; 0.75	0.74; 0.74	0.74; 0.69	0.74; 0.74	0.70; 0.71	0.71; 0.71	0.72
		Precision	0.74; 0.73	0.72; 0.73	0.74; 0.73	0.72; 0.76	0.67; 0.78	0.72; 0.76	0.69; 0.72	0.69; 0.72	0.80
		Recall	0.71; 0.77	0.71; 0.74	0.70; 0.77	0.76; 0.72	0.82; 0.62	0.77; 0.72	0.72; 0.70	0.72; 0.69	0.65
	Training set	Accuracy	0.740	0.718	0.741	0.738	0.722	0.739	0.710	0.708	0.710
		F1 score	0.74; 0.74	0.68; 0.75	0.74; 0.74	0.75; 0.73	0.73; 0.71	0.75; 0.73	0.72; 0.70	0.72; 0.70	0.66
		Precision	0.73; 0.75	0.76; 0.69	0.73; 0.75	0.71; 0.78	0.69; 0.76	0.71; 0.78	0.68; 0.75	0.68; 0.74	0.83
		Recall	0.75; 0.73	0.62; 0.81	0.75; 0.73	0.79; 0.69	0.78; 0.66	0.79; 0.69	0.76; 0.66	0.76; 0.66	0.55
	Training + Testing sets	Accuracy	0.733	0.718	0.735	0.732	0.715	0.733	0.709	0.708	0.710
		F1 score	0.73; 0.74	0.69; 0.74	0.73; 0.74	0.74; 0.72	0.73; 0.70	0.74; 0.72	0.72; 0.70	0.72; 0.70	0.67
		Precision	0.72; 0.75	0.74; 0.70	0.72; 0.75	0.70; 0.77	0.68; 0.77	0.70; 0.77	0.68; 0.74	0.68; 0.75	0.82
		Recall	0.74; 0.73	0.64; 0.79	0.74; 0.73	0.78; 0.68	0.80; 0.64	0.79; 0.68	0.75; 0.67	0.77; 0.65	0.57

Table 35. *Evaluation metrics for ML models with domain-specific words removed (training set - Spotify, testing set - Amazon)*

Appendix 4. Python Code

The code for the analysis consists of 5 different files which were reused for all the dataset combinations possible (18 files in total). The code presented here covers the code without all the possible dataset combinations. All files can be found in `https://github.com/EmiRiz26/master_thesis_code` repository.

Code for intra-domain models, example with IMDB dataset, for other datasets only path for the source and definitions of variables X,Y changed.

```
import re
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC, SVR
from sklearn.calibration import CalibratedClassifierCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix,
    precision_recall_fscore_support,mean_squared_error, mean_absolute_error,
    r2_score, roc_auc_score
from textblob import TextBlob
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from gensim.models import Word2Vec


def clean_text(text):
    if isinstance(text, str):
        text = text.lower()
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-z\s]', '', text)
        text = ' '.join(text.split())
        stop_words = set(stopwords.words('english'))
        words = text.split()
        words = [word for word in words if word not in stop_words]
        text = ' '.join(words)
        return text
```

```
    else:
        return text

def lexicon_sentiment_analysis(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

def train_word2vec(corpus):
    tokenized_corpus = [sentence.split() for sentence in corpus]
    word2vec_model = Word2Vec(sentences=tokenized_corpus, vector_size=100,
        window=5, min_count=1, workers=4)
    return word2vec_model

def vectorize_text(text, model, vector_size):
    words = text.split()
    word_vectors = [model.wv[word] for word in words if word in model.wv]
    if len(word_vectors) == 0:
        return np.zeros(vector_size)
    return np.mean(word_vectors, axis=0)

def save_sample_predictions(original_texts, sample_indices, y_true, predictions,
    model_names):
    sample_df = pd.DataFrame({
        'Text': original_texts.iloc[sample_indices].to_list(),
        'True Sentiment': y_true.iloc[sample_indices].to_list()
    })
    for model_name, preds in zip(model_names, predictions):
        sample_df[f'Predicted Sentiment ({model_name})'] = preds[sample_indices]

    sample_df.to_csv('model_sample_predictions_IMDB.csv', index=False)

def main():
    df = pd.read_csv('./IMDB_Dataset.csv')

    df['review_cleaned'] = df['review'].apply(clean_text)
    df['polarity_score'] = df['review_cleaned'].apply(lexicon_sentiment_analysis
        )

    le = LabelEncoder()
    df['sentiment_label'] = le.fit_transform(df['sentiment'])

    X = df['review_cleaned']
    y = df['sentiment_label']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
        random_state=42)

    # Polarized sentiment as regression target
    y_train_polarity = df.loc[X_train.index, 'polarity_score']
    y_test_polarity = df.loc[X_test.index, 'polarity_score']
```

```
81
82      # Vectorization
83      count_vectorizer = CountVectorizer(max_features=1000)
84      tfidf_vectorizer = TfidfVectorizer(max_features=1000)
85
86      X_train_bow = count_vectorizer.fit_transform(X_train)
87      X_test_bow = count_vectorizer.transform(X_test)
88
89      X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
90      X_test_tfidf = tfidf_vectorizer.transform(X_test)
91
92      word2vec_model = train_word2vec(X_train)
93      vector_size = 100
94
95      X_train_vec = np.array([vectorize_text(text, word2vec_model, vector_size)
            for text in X_train])
96      X_test_vec = np.array([vectorize_text(text, word2vec_model, vector_size) for
             text in X_test])
97
98
99      def evaluate_model(model, X_train, X_test, y_train, y_test, model_name,
            regression=False):
100         print(f"\nTraining {model_name}...")
101         model.fit(X_train, y_train)
102         y_pred = model.predict(X_test)
103
104         if regression:
105             mse = mean_squared_error(y_test, y_pred)
106             rmse = np.sqrt(mse)
107             mae = mean_absolute_error(y_test, y_pred)
108             r2 = r2_score(y_test, y_pred)
109             print(f"MSE: {mse:.2f}")
110             print(f"RMSE: {rmse:.2f}")
111             print(f"MAE: {mae:.2f}")
112             print(f"R^2: {r2:.2f}\n")
113         else:
114             accuracy = accuracy_score(y_test, y_pred)
115             cm = confusion_matrix(y_test, y_pred)
116             precision, recall, f1, support = precision_recall_fscore_support(
                    y_test, y_pred)
117             print(f"Accuracy: {accuracy}")
118             print(f"Confusion Matrix:\n{cm}")
119             print("\nPrecision Recall F1-Score")
120             for i in range(len(precision)):
121                 print(f"{i} {precision[i]:.2f} {recall[i]:.2f} {f1[i]:.2f} {
                        support[i]}")
122
123         return y_pred
124
```

```
    svm = CalibratedClassifierCV(LinearSVC())
    nb = MultinomialNB()
    lr = LogisticRegression(max_iter=1000, random_state=42)
    svr = SVR()

    predictions = []
    model_names = ["SVM (BOW)","Naive Bayes (BOW)", "Logistic Regression (BOW)",
                   "SVM (TF-IDF)", "Naive Bayes (TF-IDF)", "Logistic Regression
                       (TF-IDF)",
                   "SVM (Word2Vec)", "Logistic Regression (Word2Vec)", "RNN","
                       SVR (BOW)","SVR (TF-IDF)", "SVR (Word2Vec)"]

    predictions.append(evaluate_model(svm, X_train_bow, X_test_bow, y_train,
        y_test, model_names[0]))
    predictions.append(evaluate_model(nb, X_train_bow, X_test_bow, y_train,
        y_test, model_names[1]))
    predictions.append(evaluate_model(lr, X_train_bow, X_test_bow, y_train,
        y_test, model_names[2]))
    predictions.append(evaluate_model(svm, X_train_tfidf, X_test_tfidf, y_train,
         y_test, model_names[3]))
    predictions.append(evaluate_model(nb, X_train_tfidf, X_test_tfidf, y_train,
        y_test, model_names[4]))
    predictions.append(evaluate_model(lr, X_train_tfidf, X_test_tfidf, y_train,
        y_test, model_names[5]))
    predictions.append(evaluate_model(svm, X_train_vec, X_test_vec, y_train,
        y_test, model_names[6]))
    predictions.append(evaluate_model(lr, X_train_vec, X_test_vec, y_train,
        y_test, model_names[7]))
    predictions.append(evaluate_model(svr, X_train_bow, X_test_bow,
        y_train_polarity, y_test_polarity, model_names[9], regression=True))
    predictions.append(evaluate_model(svr, X_train_tfidf, X_test_tfidf,
        y_train_polarity, y_test_polarity, model_names[10], regression=True))
    predictions.append(evaluate_model(svr, X_train_vec, X_test_vec,
        y_train_polarity, y_test_polarity, model_names[11], regression=True))


    # RNN model
    tokenizer = Tokenizer(num_words=10000)
    tokenizer.fit_on_texts(X_train)
    X_train_seq = tokenizer.texts_to_sequences(X_train)
    X_test_seq = tokenizer.texts_to_sequences(X_test)
    X_train_pad = pad_sequences(X_train_seq, maxlen=100)
    X_test_pad = pad_sequences(X_test_seq, maxlen=100)

    model_rnn = Sequential([
    Embedding(input_dim=10000, output_dim=256),
    LSTM(128, return_sequences=True),
    Dropout(0.2),
    LSTM(128),
```

```
    Dense(1, activation='sigmoid')
    ])
    model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
        accuracy'])

    print("\nTraining RNN...")
    model_rnn.fit(X_train_pad, y_train, epochs=3, batch_size=64,
        validation_split=0.2)

    rnn_pred = model_rnn.predict(X_test_pad).flatten()
    rnn_pred_label = (rnn_pred > 0.5).astype(int)
    predictions.append(rnn_pred_label)

    sample_indices = np.random.choice(range(len(X_test)), 10, replace=False)
    save_sample_predictions(X, sample_indices, y, predictions, model_names)

    rnn_accuracy = accuracy_score(y_test, rnn_pred_label)
    rnn_precision, rnn_recall, rnn_f1, _ = precision_recall_fscore_support(
        y_test, rnn_pred_label, average='binary')
    rnn_auc = roc_auc_score(y_test, rnn_pred)

    print(f"RNN model accuracy: {rnn_accuracy:.2f}")
    print(f"RNN model precision: {rnn_precision:.2f}")
    print(f"RNN model recall: {rnn_recall:.2f}")
    print(f"RNN model F1 Score: {rnn_f1:.2f}")
    print(f"RNN model ROC AUC Score: {rnn_auc:.2f}")

if __name__ == "__main__":
    main()
...
```

Listing 1: Script for IMDB intra-domain model

Code for cross-domain models, example with IMDB on Twitter model, for other datasets only paths for the sources and definitions of variables X,Y changed, together with different lists of domain specific words.

```
import re
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC, SVR
from sklearn.calibration import CalibratedClassifierCV
from sklearn.preprocessing import LabelEncoder
```

```
from sklearn.metrics import accuracy_score, confusion_matrix,
    precision_recall_fscore_support,mean_squared_error, mean_absolute_error,
    r2_score, roc_auc_score
from textblob import TextBlob
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from gensim.models import Word2Vec


def clean_text(text, domain_specific_words=None):
    if isinstance(text, str):
        text = text.lower()
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-z\s]', '', text)
        text = ' '.join(text.split())
        stop_words = set(stopwords.words('english'))
        if domain_specific_words:
            stop_words = stop_words.union(set(domain_specific_words))
        words = text.split()
        words = [word for word in words if word not in stop_words]
        text = ' '.join(words)
        return text
    else:
        return text

def lexicon_sentiment_analysis(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

def train_word2vec(corpus):
    tokenized_corpus = [sentence.split() for sentence in corpus]
    word2vec_model = Word2Vec(sentences=tokenized_corpus, vector_size=100,
        window=5, min_count=1, workers=4)
    return word2vec_model

def vectorize_text(text, model, vector_size):
    words = text.split()
    word_vectors = [model.wv[word] for word in words if word in model.wv]
    if len(word_vectors) == 0:
        return np.zeros(vector_size)
    return np.mean(word_vectors, axis=0)

def save_sample_predictions(original_texts, sample_indices, y_true, predictions,
    model_names):
    sample_df = pd.DataFrame({
```

```
        'Text': original_texts.iloc[sample_indices].to_list(),
        'True Sentiment': y_true.iloc[sample_indices].to_list()
    })
    for model_name, preds in zip(model_names, predictions):
        sample_df[f'Predicted Sentiment ({model_name})'] = preds[sample_indices]

    sample_df.to_csv('model_sample_predictions_IMDB_Twitter.csv', index=False)

def main():

    domain_specific_words_twitter = [
    # Common Twitter slang & abbreviations
    "rt", "retweet", "dm", "fomo", "idk", "lol", "lmao", "rofl", "brb",
    "omg", "wtf", "btw", "tbh", "imo", "imho", "irl", "smh",
    "afaik", "ftw", "ftl", "dm", "tl", "bf", "bff",

    # Common hashtags & mentions (if you want to remove)
    "hashtag", "follow", "followers", "followback", "followfriday", "ff",
    "mention", "mentions", "like", "likes", "reply", "replies",

    # Profanity and mild curse words (use with caution, adjust based on context)
    "shit", "damn", "fuck", "fucking", "bitch", "bastard", "crap", "asshole",
    "dick", "piss", "bollocks", "bloody", "bugger", "arse", "cock", "cunt",

    # Common Twitter phrases & filler words
    "yo", "hey", "sup", "wanna", "gonna", "gotta", "kinda", "lemme", "ain't",
    "gotta", "lemme", "cuz", "cause", "tho", "thx", "pls", "plz",

    # Other filler or frequent terms
    "tweet", "tweets", "twitter", "timeline", "trending", "viral", "trends",

    # URL or mentions placeholders (optional if you remove these before)
    "http", "https", "www", "com", "co", "twittercom", "tco"
]
    domain_specific_words_imdb = [
    # Genres
    "action", "comedy", "drama", "horror", "thriller", "romance", "sci-fi", "
        documentary", "animation",

    # Roles
    "director", "actor", "actress", "producer", "writer", "filmmaker", "cast", "
        crew",

    # Cinematic Elements
    "screenplay", "dialogue", "plot", "character", "scene", "climax", "
        soundtrack", "special effects", "cinematography",

    # Releases and Viewings
```

```
    "premiere", "release", "screening", "sequel", "prequel", "trilogy", "
        adaptation",

    # Technical Terms
    "editing", "score", "sound design", "visual effects", "production", "
        lighting",

    # Common Review Phrases specific to film context
    #"masterpiece", "overrated", "underrated", "classic", "disappointing", "
        lackluster",

    # Emotions and Reactions relevant to films
    #"funny", "scary", "thrilling", "boring", "touching", "emotional", "engaging
        ", "entertaining",

    # Audience and Viewing Context
    "viewer", "audience", "fans", "fandom", "blockbuster", "box office", "flop",
    "theater", "cinema", "home viewing", "matinee", "midnight screening"
]
    df_IMDB = pd.read_csv('./IMDB_Dataset.csv')

    # df_IMDB['review_cleaned'] = df_IMDB['review'].apply(clean_text)
    df_IMDB['review_cleaned'] = df_IMDB['review'].apply(lambda x: clean_text(x,
        domain_specific_words_imdb)).fillna('')
    df_IMDB['polarity_score'] = df_IMDB['review_cleaned'].apply(
        lexicon_sentiment_analysis)

    le = LabelEncoder()
    df_IMDB['sentiment_label'] = le.fit_transform(df_IMDB['sentiment'])

    df_Twitter = pd.read_csv('./twitter_training.csv', header=None)

    # Ensure that the DataFrame has at least 3 columns (adjust the numbers if
        incorrect)
    if df_Twitter.shape[1] > 2:
        # Filter to only keep rows where sentiment is "Positive" or "Negative"
        df_Twitter = df_Twitter[df_Twitter[2].isin(['Positive', 'Negative'])]

        # Clean the text in column 3 and handle NaN by filling with an empty
            string
        # df_Twitter['review_cleaned'] = df_Twitter[3].apply(clean_text).fillna
            ('')
        df_Twitter['review_cleaned'] = df_Twitter[3].apply(lambda x: clean_text(
            x, domain_specific_words_twitter)).fillna('')

        # Encode "Positive" as 1 and "Negative" as 0
        le = LabelEncoder()
        df_Twitter['sentiment_label'] = le.fit_transform(df_Twitter[2])  #
            Encode the sentiments
```

```
    else:
        print("Error: DataFrame does not have the expected number of columns.")

    df_Twitter['polarity_score'] = df_Twitter['review_cleaned'].apply(
        lexicon_sentiment_analysis)

    X_test = df_Twitter['review_cleaned']
    y_test = df_Twitter['sentiment_label']

    X_train = df_IMDB['review_cleaned']
    y_train = df_IMDB['sentiment_label']

    # Polarized sentiment as regression target
    y_train_polarity = df_IMDB['polarity_score']
    y_test_polarity = df_Twitter['polarity_score']

    # Vectorization
    count_vectorizer = CountVectorizer(max_features=1000)
    tfidf_vectorizer = TfidfVectorizer(max_features=1000)

    X_train_bow = count_vectorizer.fit_transform(X_train)
    X_test_bow = count_vectorizer.transform(X_test)

    X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
    X_test_tfidf = tfidf_vectorizer.transform(X_test)

    word2vec_model = train_word2vec(X_train)
    vector_size = 100

    X_train_vec = np.array([vectorize_text(text, word2vec_model, vector_size)
        for text in X_train])
    X_test_vec = np.array([vectorize_text(text, word2vec_model, vector_size) for
         text in X_test])


    def evaluate_model(model, X_train, X_test, y_train, y_test, model_name,
        regression=False):
        print(f"\nTraining {model_name}...")
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        if regression:
            mse = mean_squared_error(y_test, y_pred)
            rmse = np.sqrt(mse)
            mae = mean_absolute_error(y_test, y_pred)
            r2 = r2_score(y_test, y_pred)
            print(f"MSE: {mse:.2f}")
            print(f"RMSE: {rmse:.2f}")
            print(f"MAE: {mae:.2f}")
```

```
            print(f"R^2: {r2:.2f}\n")
        else:
            accuracy = accuracy_score(y_test, y_pred)
            cm = confusion_matrix(y_test, y_pred)
            precision, recall, f1, support = precision_recall_fscore_support(
                y_test, y_pred)
            print(f"Accuracy: {accuracy}")
            print(f"Confusion Matrix:\n{cm}")
            print("\nPrecision Recall F1-Score")
            for i in range(len(precision)):
                print(f"{i} {precision[i]:.2f} {recall[i]:.2f} {f1[i]:.2f} {
                    support[i]}")

        return y_pred

    svm = CalibratedClassifierCV(LinearSVC())
    nb = MultinomialNB()
    lr = LogisticRegression(max_iter=1000, random_state=42)
    svr = SVR()

    predictions = []
    model_names = ["SVM (BOW)","Naive Bayes (BOW)", "Logistic Regression (BOW)",
                   "SVM (TF-IDF)", "Naive Bayes (TF-IDF)", "Logistic Regression
                       (TF-IDF)",
                   "SVM (Word2Vec)", "Logistic Regression (Word2Vec)", "RNN","
                       SVR (BOW)","SVR (TF-IDF)", "SVR (Word2Vec)"]

    predictions.append(evaluate_model(svm, X_train_bow, X_test_bow, y_train,
        y_test, model_names[0]))
    predictions.append(evaluate_model(nb, X_train_bow, X_test_bow, y_train,
        y_test, model_names[1]))
    predictions.append(evaluate_model(lr, X_train_bow, X_test_bow, y_train,
        y_test, model_names[2]))
    predictions.append(evaluate_model(svm, X_train_tfidf, X_test_tfidf, y_train,
         y_test, model_names[3]))
    predictions.append(evaluate_model(nb, X_train_tfidf, X_test_tfidf, y_train,
        y_test, model_names[4]))
    predictions.append(evaluate_model(lr, X_train_tfidf, X_test_tfidf, y_train,
        y_test, model_names[5]))
    predictions.append(evaluate_model(svm, X_train_vec, X_test_vec, y_train,
        y_test, model_names[6]))
    predictions.append(evaluate_model(lr, X_train_vec, X_test_vec, y_train,
        y_test, model_names[7]))
    predictions.append(evaluate_model(svr, X_train_bow, X_test_bow,
        y_train_polarity, y_test_polarity, model_names[9], regression=True))
    predictions.append(evaluate_model(svr, X_train_tfidf, X_test_tfidf,
        y_train_polarity, y_test_polarity, model_names[10], regression=True))
    predictions.append(evaluate_model(svr, X_train_vec, X_test_vec,
        y_train_polarity, y_test_polarity, model_names[11], regression=True))
```

```
221
222     sample_indices = np.random.choice(len(X_test), 10, replace=False)
223     save_sample_predictions(X_test, sample_indices, y_test, predictions,
            model_names)
224
225     # RNN model
226     tokenizer = Tokenizer(num_words=10000)
227     tokenizer.fit_on_texts(X_train)
228     X_train_seq = tokenizer.texts_to_sequences(X_train)
229     X_test_seq = tokenizer.texts_to_sequences(X_test)
230     X_train_pad = pad_sequences(X_train_seq, maxlen=100)
231     X_test_pad = pad_sequences(X_test_seq, maxlen=100)
232
233     model_rnn = Sequential([
234     Embedding(input_dim=10000, output_dim=256),
235     LSTM(128, return_sequences=True),
236     Dropout(0.2),
237     LSTM(128),
238     Dense(1, activation='sigmoid')
239     ])
240     model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
            accuracy'])
241
242     print("\nTraining RNN...")
243     model_rnn.fit(X_train_pad, y_train, epochs=3, batch_size=64,
            validation_split=0.2)
244
245     rnn_pred = model_rnn.predict(X_test_pad).flatten()
246     rnn_pred_label = (rnn_pred > 0.5).astype(int)
247     predictions.append(rnn_pred_label)
248
249     rnn_accuracy = accuracy_score(y_test, rnn_pred_label)
250     rnn_precision, rnn_recall, rnn_f1, _ = precision_recall_fscore_support(
            y_test, rnn_pred_label, average='binary')
251     rnn_auc = roc_auc_score(y_test, rnn_pred)
252
253     print(f"RNN model accuracy: {rnn_accuracy:.2f}")
254     print(f"RNN model precision: {rnn_precision:.2f}")
255     print(f"RNN model recall: {rnn_recall:.2f}")
256     print(f"RNN model F1 Score: {rnn_f1:.2f}")
257     print(f"RNN model ROC AUC Score: {rnn_auc:.2f}")
258
259 if __name__ == "__main__":
260     main()
```

Listing 2: Script for cross-domain models

Code for models with combined training datasets.

```
1 import re
```

```
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
from textblob import TextBlob
from gensim.models import Word2Vec
from sklearn.ensemble import VotingClassifier

def clean_text(text, domain_specific_words=None):
    if isinstance(text, str):
        text = text.lower()
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-z\s]', '', text)
        text = ' '.join(text.split())
        stop_words = set(stopwords.words('english'))
        if domain_specific_words:
            stop_words = stop_words.union(set(domain_specific_words))
        words = text.split()
        words = [word for word in words if word not in stop_words]
        text = ' '.join(words)
        return text
    else:
        return text

def lexicon_sentiment_analysis(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

def train_word2vec(corpus):
    tokenized_corpus = [sentence.split() for sentence in corpus]
    word2vec_model = Word2Vec(sentences=tokenized_corpus, vector_size=100,
        window=8, min_count=1, workers=4)
    return word2vec_model

def vectorize_text(text, model, vector_size):
    words = text.split()
    word_vectors = [model.wv[word] for word in words if word in model.wv]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(
        vector_size)

```

```python
def evaluate(models, X_train, y_train, X_test, y_test, test_name, prediction_output, metrics_output):
    prediction_df = pd.DataFrame({'True Label': y_test})
    metrics_list = []

    for model_name, model, X_train_feature, X_test_feature in models:
        print(f"\nEvaluating {model_name} on {test_name}...")
        model.fit(X_train_feature, y_train)
        predictions = model.predict(X_test_feature)

        accuracy = accuracy_score(y_test, predictions)
        precision, recall, f1, _ = precision_recall_fscore_support(y_test, predictions, average='binary')
        print(f"Accuracy: {accuracy}, Precision: {precision}, Recall: {recall}, F1 Score: {f1}")

        prediction_df[model_name] = predictions

        metrics_dict = {
            'Model': model_name,
            'Accuracy': accuracy,
            'Precision': precision,
            'Recall': recall,
            'F1 Score': f1,
        }
        metrics_list.append(metrics_dict)

    # Ensemble for each feature type
    feature_models = {
        "BOW": [m for m in models if 'BOW' in m[0]],
        "TFIDF": [m for m in models if 'TF-IDF' in m[0]],
        "Word2Vec": [m for m in models if 'Word2Vec' in m[0]],
    }

    for feature_type, feature_models in feature_models.items():
        if not feature_models:
            continue

        ensemble = VotingClassifier(estimators=[
            (name, model) for name, model, _, _ in feature_models
        ], voting='hard')

        train_feature = feature_models[0][2]  # Training feature
        test_feature = feature_models[0][3]   # Testing feature

        ensemble.fit(train_feature, y_train)
        ensemble_predictions = ensemble.predict(test_feature)

        ensemble_accuracy = accuracy_score(y_test, ensemble_predictions)
```

```
        ensemble_precision, ensemble_recall, ensemble_f1, _ =
            precision_recall_fscore_support(y_test, ensemble_predictions, average
            ='binary')
        print(f"\n{feature_type} Feature Ensemble Model Accuracy: {
            ensemble_accuracy}")

        prediction_df[f'{feature_type} Ensemble'] = ensemble_predictions

        metrics_list.append({
            'Model': f'{feature_type} Ensemble',
            'Accuracy': ensemble_accuracy,
            'Precision': ensemble_precision,
            'Recall': ensemble_recall,
            'F1 Score': ensemble_f1,
        })

    metrics_df = pd.DataFrame(metrics_list)

    prediction_df.to_csv(prediction_output, index=False)
    metrics_df.to_csv(metrics_output, index=False)

def main():
    datasets = {
        'IMDB': './IMDB_Dataset.csv',
        'Spotify': './Spotify_app.csv',
        'Amazon': './Amazon.csv',
        'RT': './Rotten_tomatoes.csv',
        'Twitter': './twitter_training.csv'
    }

    # Prepare all datasets
    dfs = {}
    for name, path in datasets.items():
        if name == 'Amazon':
            df = pd.read_csv(path, header=None, nrows=55000)
            df['review_cleaned'] = df[2].apply(clean_text).fillna('')
            df[0] = df[0].map({1: 0, 2: 1})
            df['sentiment_label'] = df[0]
        elif name == 'Twitter':
            df = pd.read_csv(path, header=None)
            df = df[df[2].isin(['Positive', 'Negative'])]
            df['review_cleaned'] = df[3].apply(clean_text).fillna('')
            le = LabelEncoder()
            df['sentiment_label'] = le.fit_transform(df[2])
        elif name == 'IMDB':
            df = pd.read_csv(path)
            df['review_cleaned'] = df['review'].apply(clean_text).fillna('')
            le = LabelEncoder()
            df['sentiment_label'] = le.fit_transform(df['sentiment'])
```

```
        elif name == 'Spotify':
            df = pd.read_csv(path)
            df['review_cleaned'] = df['Review'].apply(clean_text).fillna('')
            le = LabelEncoder()
            df['sentiment_label'] = le.fit_transform(df['label'])
        else:  # Rotten tomatoes
            df = pd.read_csv(path)
            df['review_cleaned'] = df['text'].apply(clean_text).fillna('')
            df['sentiment_label'] = df['label']

        dfs[name] = df

    # Loop over datasets to use each as a test set once
    for test_name in datasets.keys():
        print(f"\nRunning evaluation with {test_name} as the test set...")

        # Create test set
        df_test = dfs[test_name]
        X_test = df_test['review_cleaned']
        y_test = df_test['sentiment_label']

        # Create training set by combining other datasets
        train_dfs = [df for key, df in dfs.items() if key != test_name]
        df_train = pd.concat(train_dfs)

        X_train = df_train['review_cleaned']
        y_train = df_train['sentiment_label']

        # Vectorization
        count_vectorizer = CountVectorizer(max_features=1000)
        tfidf_vectorizer = TfidfVectorizer(max_features=1000)

        X_train_bow = count_vectorizer.fit_transform(X_train)
        X_test_bow = count_vectorizer.transform(X_test)

        X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
        X_test_tfidf = tfidf_vectorizer.transform(X_test)

        word2vec_model = train_word2vec(X_train)
        vector_size = 100

        X_train_vec = np.array([vectorize_text(text, word2vec_model, vector_size
            ) for text in X_train])
        X_test_vec = np.array([vectorize_text(text, word2vec_model, vector_size)
             for text in X_test])

        # Define models
        models = [
            ("SVM (BOW)", CalibratedClassifierCV(LinearSVC()), X_train_bow),
```

```
            ("Naive Bayes (BOW)", MultinomialNB(), X_train_bow),
            ("Logistic Regression (BOW)", LogisticRegression(max_iter=1000),
                X_train_bow),
            ("SVM (TF-IDF)", CalibratedClassifierCV(LinearSVC()), X_train_tfidf)
                ,
            ("Naive Bayes (TF-IDF)", MultinomialNB(), X_train_tfidf),
            ("Logistic Regression (TF-IDF)", LogisticRegression(max_iter=1000),
                X_train_tfidf),
            ("SVM (Word2Vec)", CalibratedClassifierCV(LinearSVC()), X_train_vec)
                ,
            ("Logistic Regression (Word2Vec)", LogisticRegression(max_iter=1000)
                , X_train_vec),
        ]

        # Combine models and test features
        models_eval = [
            (name, model, train_feature, test_feature)
            for (name, model, train_feature), test_feature in zip(models, [
                X_test_bow, X_test_bow, X_test_bow, X_test_tfidf, X_test_tfidf,
                X_test_tfidf, X_test_vec, X_test_vec])
        ]

        # File paths for output
        prediction_output = f"{test_name}_predictions.csv"
        metrics_output = f"{test_name}_metrics.csv"

        # Evaluate models on the current test set
        evaluate(models_eval, X_train, y_train, X_test, y_test, test_name,
            prediction_output, metrics_output)

if __name__ == "__main__":
    main()
```

Listing 3: Script for models with combined training datasets

Code for ensemble averaging.

```
import re
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.calibration import CalibratedClassifierCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
```

```
from sklearn.ensemble import VotingClassifier
from textblob import TextBlob
from gensim.models import Word2Vec


def clean_text(text, domain_specific_words=None):
    if isinstance(text, str):
        text = text.lower()
        text = re.sub(r'<.*?>', '', text)
        text = re.sub(r'[^a-z\s]', '', text)
        text = ' '.join(text.split())
        stop_words = set(stopwords.words('english'))
        if domain_specific_words:
            stop_words = stop_words.union(set(domain_specific_words))
        words = text.split()
        words = [word for word in words if word not in stop_words]
        text = ' '.join(words)
        return text
    else:
        return text

def lexicon_sentiment_analysis(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

def train_word2vec(corpus):
    tokenized_corpus = [sentence.split() for sentence in corpus]
    word2vec_model = Word2Vec(sentences=tokenized_corpus, vector_size=100,
        window=8, min_count=1, workers=4)
    return word2vec_model

def vectorize_text(text, model, vector_size):
    words = text.split()
    word_vectors = [model.wv[word] for word in words if word in model.wv]
    return np.mean(word_vectors, axis=0) if word_vectors else np.zeros(
        vector_size)

def evaluate_ensemble(models, X_train, y_train, X_test, y_test, test_name,
    prediction_output, metrics_output):
    # Creating a Voting Classifier
    voting_classifier = VotingClassifier(estimators=[
        (name, model) for name, model, _, _ in models
    ], voting='soft')  # 'soft' voting averages class probabilities

    # Fit the ensemble model on the centralized feature space
    print(f"\nTraining ensemble on all feature spaces for {test_name}...")
    voting_classifier.fit(np.hstack([X_train_feature for _, _, X_train_feature,
        _ in models]), y_train)

```

```
    # Make predictions on the test set
    predictions = voting_classifier.predict(np.hstack([X_test_feature for _, _,
        _, X_test_feature in models]))

    # Evaluate predictions
    accuracy = accuracy_score(y_test, predictions)
    precision, recall, f1, _ = precision_recall_fscore_support(y_test,
        predictions, average='binary')
    print(f"\nEnsemble Results: Accuracy: {accuracy}, Precision: {precision},
        Recall: {recall}, F1 Score: {f1}")

    # Save predictions to a DataFrame
    prediction_df = pd.DataFrame({
        'True Label': y_test,
        'Ensemble Prediction': predictions
    })

    metrics_dict = {
        'Model': 'Ensemble',
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1 Score': f1,
    }

    # Save results to CSV
    prediction_df.to_csv(prediction_output, index=False)
    metrics_df = pd.DataFrame([metrics_dict])
    metrics_df.to_csv(metrics_output, index=False)

def main():
    datasets = {
        'IMDB': './IMDB_Dataset.csv',
        'Spotify': './Spotify_app.csv',
        'Amazon': './Amazon.csv',
        'RT': './Rotten_tomatoes.csv',
        'Twitter': './twitter_training.csv'
    }

    # Prepare all datasets
    dfs = {}
    for name, path in datasets.items():
        if name == 'Amazon':
            df = pd.read_csv(path, header=None, nrows=55000)
            df['review_cleaned'] = df[2].apply(clean_text).fillna('')
            df[0] = df[0].map({1: 0, 2: 1})
            df['sentiment_label'] = df[0]
        elif name == 'Twitter':
            df = pd.read_csv(path, header=None)
```

```
            df = df[df[2].isin(['Positive', 'Negative'])]
            df['review_cleaned'] = df[3].apply(clean_text).fillna('')
            le = LabelEncoder()
            df['sentiment_label'] = le.fit_transform(df[2])
        elif name == 'IMDB':
            df = pd.read_csv(path)
            df['review_cleaned'] = df['review'].apply(clean_text).fillna('')
            le = LabelEncoder()
            df['sentiment_label'] = le.fit_transform(df['sentiment'])
        elif name == 'Spotify':
            df = pd.read_csv(path)
            df['review_cleaned'] = df['Review'].apply(clean_text).fillna('')
            le = LabelEncoder()
            df['sentiment_label'] = le.fit_transform(df['label'])
        else:  # Rotten tomatoes
            df = pd.read_csv(path)
            df['review_cleaned'] = df['text'].apply(clean_text).fillna('')
            df['sentiment_label'] = df['label']

        dfs[name] = df

    for test_name in datasets.keys():
        print(f"\nRunning ensemble evaluation with {test_name} as the test set
            ...")

        # Create test set
        df_test = dfs[test_name]
        X_test = df_test['review_cleaned']
        y_test = df_test['sentiment_label']

        # Create training set by combining other datasets
        train_dfs = [df for key, df in dfs.items() if key != test_name]
        df_train = pd.concat(train_dfs)

        X_train = df_train['review_cleaned']
        y_train = df_train['sentiment_label']

        # Vectorization
        count_vectorizer = CountVectorizer(max_features=1000)
        tfidf_vectorizer = TfidfVectorizer(max_features=1000)

        X_train_bow = count_vectorizer.fit_transform(X_train)
        X_test_bow = count_vectorizer.transform(X_test)

        X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
        X_test_tfidf = tfidf_vectorizer.transform(X_test)

        word2vec_model = train_word2vec(X_train)
        vector_size = 100
```

```
153
154  # Convert 1D Word2Vec vectors to 2D
155          X_train_vec = np.array([vectorize_text(text, word2vec_model, vector_size
                 ) for text in X_train])
156          X_test_vec = np.array([vectorize_text(text, word2vec_model, vector_size)
                  for text in X_test])
157
158          # Reshape to 2D arrays: (n_samples, n_features)
159          X_train_vec = X_train_vec.reshape(-1, vector_size)
160          X_test_vec = X_test_vec.reshape(-1, vector_size)
161
162          # Define models
163          models = [
164              ("SVM (BOW)", CalibratedClassifierCV(LinearSVC()), X_train_bow),
165              ("Naive Bayes (BOW)", MultinomialNB(), X_train_bow),
166              ("Logistic Regression (BOW)", LogisticRegression(max_iter=1000),
                     X_train_bow),
167              ("SVM (TF-IDF)", CalibratedClassifierCV(LinearSVC()), X_train_tfidf)
                     ,
168              ("Naive Bayes (TF-IDF)", MultinomialNB(), X_train_tfidf),
169              ("Logistic Regression (TF-IDF)", LogisticRegression(max_iter=1000),
                     X_train_tfidf),
170              ("SVM (Word2Vec)", CalibratedClassifierCV(LinearSVC()), X_train_vec)
                     ,
171              ("Logistic Regression (Word2Vec)", LogisticRegression(max_iter=1000)
                     , X_train_vec),
172          ]
173
174          # Combine models and test features
175          models_eval = [
176              (name, model, train_feature, test_feature)
177              for (name, model, train_feature), test_feature in zip(models, [
                     X_test_bow, X_test_bow, X_test_bow, X_test_tfidf, X_test_tfidf,
                     X_test_tfidf, X_test_vec, X_test_vec])
178          ]
179
180          # File paths for output
181          prediction_output = f"{test_name}_predictions_ensemble.csv"
182          metrics_output = f"{test_name}_metrics_ensemble.csv"
183
184          # Evaluate ensemble
185          evaluate_ensemble(models_eval, X_train, y_train, X_test, y_test,
                 test_name, prediction_output, metrics_output)
186
187  if __name__ == "__main__":
188      main()
```

Listing 4: Script for ensemble averaging

Code used in a notebook to conduct Friedman and Nemenyi statistical tests, example with Accuracy scores for models tested on Amazon dataset, for other datasets, only path of the source changed.

```
import pandas as pd
import numpy as np
from scipy import stats
import scikit_posthocs as sp # https://pypi.org/project/scikit-posthocs/
import stac
import matplotlib.pyplot as plt

# Assumes a formattting where rows are datasets, and the methods are columns
df = pd.read_csv("./Amazon - Accuracy.csv", index_col=0)
# df = df.drop(df.columns[-1], axis=1)
# df = df.applymap(lambda x: float(x.split(';')[1].strip()))
data = np.asarray(df)

print("Read data")

# To be safe, ensure this matches what was expected
num_datasets, num_methods = data.shape
print("Methods:", num_methods, "Datasets:", num_datasets)

alpha = 0.05 # Set this to the desired alpha/signifance level

stat, p = stats.friedmanchisquare(*data)

reject = p <= alpha
print("Should we reject H0 (i.e. is there a difference in the means) at the",
    (1-alpha)*100, "% confidence level?", reject)

# Helper functions for performing the statistical tests
def generate_scores(method, method_args, data, labels):
    pairwise_scores = method(data, **method_args) # Matrix for all pairwise
        comaprisons
    pairwise_scores.set_axis(labels, axis='columns', inplace=True) # Label the
        cols
    pairwise_scores.set_axis(labels, axis='rows', inplace=True) # Label the rows
        , note: same label as pairwise combinations
    return pairwise_scores

def plot(scores):
    # Pretty plot of significance
    heatmap_args = {'linewidths': 0.25, 'linecolor': '0.5', 'square': True,
                    'cbar_ax_bbox': [0.80, 0.35, 0.04, 0.3]}

    sp.sign_plot(scores, **heatmap_args)

    def generate_scores(method, method_args, data, labels):
```

```
42      pairwise_scores = method(data, **method_args)  # Matrix for all pairwise
            comparisons
43      pairwise_scores.columns = labels  # Label the columns
44      pairwise_scores.index = labels    # Label the rows, same label as pairwise
            combinations
45      return pairwise_scores
46  nemenyi_scores = generate_scores(sp.posthoc_nemenyi_friedman, {}, data, df.
        columns)
47
48  plot(nemenyi_scores)
49  plt.suptitle("Accuracy p-values (testing set - Amazon)")
50  plt.show()
```

Listing 5: Script for statistical tests notebook