WIRTSCHAFTSUNIVERSITÄT WIEN
Vienna University of Economics and Business

Bachelor Thesis

Title of Bachelor Thesis (english)	Comparing different ML Algorithms on Sentiment Analysis and Genre Classification of Steam Reviews
Title of Bachelor Thesis (german)	-
Author (last name, first name):	Krizanic Sebastian
Student ID number:	h01639381
Degree program:	Bachelor of Science (WU), BSc (WU)
Examiner (degree, first name, last name):	Prof. Mitlöhner Johann

I hereby declare that:

1. I have written this Bachelor thesis myself, independently and without the aid of unfair or unauthorized resources. Whenever content has been taken directly or indirectly from other sources, this has been indicated and the source referenced.

2. This Bachelor Thesis has not been previously presented as an examination paper in this or any other form in Austria or abroad.

3. This Bachelor Thesis is identical with the thesis assessed by the examiner.

4. (only applicable if the thesis was written by more than one author): this Bachelor thesis was written together with

   The individual contributions of each writer as well as the co-written passages have been indicated.

10/04/2024
Date

Unterschrift

WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

Bachelor Thesis

Comparing different ML Algorithms on Sentiment Analysis and Genre Classification of Steam Reviews

Krizanic Sebastian

Date of Birth: 21.02.1997
Student ID: 01639381

Subject Area: Data Science

Studienkennzahl: J 033 561

Supervisor: Mitloehner Johann

Date of Submission: 18th June 2024

Department of Information Systems & Operations Management, Vienna University of Economics and Business, Welthandelsplatz 1, 1020 Vienna, Austria

DEPARTMENT FÜR WIRTSCHAFTSINFORMATIK
UND OPERATIONS MANAGEMENT DEPARTMENT
OF INFORMATION SYSTEMS AND OPERATIONS
MANAGEMENT

Contents

List of Figures

List of Tables

Abstract

Machine Learning allows for efficient classification of text for various purposes. This thesis focuses on the performance of certain algorithms being used on video game reviews, specifically, reviews from the online PC Video Game retailer Steam. Steam allows to extract reviews from their database, which means simple access to a large amount of labeled data. The goal of this thesis is to test various machine learning algorithms on Sentiment Analysis and Genre Classification (or Multi-class classification). The performance will be measured by accuracy, as well as Confusion Matrices. The goal has been reached by being able to use Steams API for sufficient amount of data. Additional to the methods mentioned above, grouped bar plots and performance tables have been utilized to present the results in an intuitive manner as well as a brief analysis of the results. The results show an average accuracy for sentiment analysis of 88 percent, while for genre classification it averaged around 69 percent. These values stem from models where no subset had to be used.

1 Introduction

Gaming is an ever-growing sector, the revenue rivaling even the movie and music industry [35]. Thanks to its huge popularity, online video game retailers are thriving, especially this particular platform: https://store.steampowered.com/. Steam was brought to life by Valve in order to give players an easy way to keep their games updated. As the years passed, the platform developed into a retail platform for PC Games, generating not only a lot of revenue, but also a significant amount of data, which can easily be accessed by using the provided Steam API. Thanks to this policy, getting a large amount of labeled data poses no problem.

The goal is to gather recent reviews about a variety of games in order to determine if certain machine learning algorithms are able to correctly predict the sentiment as well as the genre of the game. Afterwards, the performance scores will be visualized and evaluated.

1.1 Research question

1. How do various machine learning algorithms perform when classifying the sentiment and genre of steam reviews?
2. How does the performance of these algorithms compare to each other? Is there a significant difference between performance on sentiment and multiclassing problems?
3. Is there a significant difference between classic machine learning and neural networks?
4. How does multi-class classification perform when specific keywords are being filtered out? (Genre tags e.g.)

1.2 Research method

The research method is gathering reviews from the steam database, cleaning them and finally using machine learning algorithms on those steam reviews to gather performance variables in order to evaluate the aforementioned algorithms. Confusion matrices as well as accuracy scores will be used to compare all chosen algorithms, while for specific methods other suitable graphs may be chosen in addition. Additionally, performance tables and grouped bar plots will be used as well as brief analyses.

2 Collection, Cleaning and Distribution of the Steam Reviews

2.1 Collecting Steam Reviews by API

All of the code can be found in the following github repository: https://github.com/F34R7/Comparing-different-ML-Algorithms-on-Sentiment-Analysis-and-Genre-Classification-of-Steam-Reviews[33].

In order to adhere to the rules, please consult https://store.steampowered.com/subscriber_agreement/#:~:text=You%20are%20entitled%20to%20use,others%20without%20the%20prior%20written. The scraped data is being used for educational purposes.

The first step is to gather steam reviews. In order to know which games belong to which ID, the website [1] https://steamdb.info/ will be used. It is a database with all games on Steam, as well as additional data like player count etc., which are not relevant for this thesis. Furthermore, this website [2] https://partner.steamgames.com/doc/store/getreviews explains how to use the Steam API in order to scrape for reviews.

In order to generate a valid API request, a valid game_id is needed, which can be found in the steam database. The language can be filtered to any language supported by steam. The filter allows to further narrow down the search, which for this thesis the interest lies in recent reviews. The cursor variable is used to track how many reviews were scraped in order to prevent duplicates. The cursor variable is being updated after each call of the API. Each iteration of the for loop yields a batch of 20 reviews. The reviews in this state only contain the actual review text as well as the sentiment, but not which genre the game belongs to. This is added manually for each game which is scraped by looking at the tags on the official steam page for the corresponding game. This means that the genre is not necessarily correct in the sense that a first person shooter might also be a coop game. This was also considered when choosing which games ought to be included in this thesis.

Once the data has been labeled correctly, it is added to a csv file ready for the cleaning process to commence. A further note: Within the jupyter notebook the exact reviews that were scraped are persisted in a markdown file. It contains the exact game name, id, as well as the genre and the amount of reviews scraped in addition to the date of scraping. This was done to ensure transparency.

2.2 Cleaning the Steam Reviews

The first step is to remove special characters from the reviews because they do not add any value to the algorithms.

The second step consists of removing stopwords as well as lemmatization in order to boost performance. NLTK was used for these steps. After successfully cleaning the reviews, they get tokenized.

[1] The steam database website: [38]
[2] API Description: [39]

Following this step, a second CSV will be generated. This specific CSV will have all genre specific tags removed in order to evaluate if the model can still correctly predict the game genre without these keywords.

With these steps accomplished, the dataset is ready to be trained on.

2.3 Distribution of the Review Classes

The distribution of the reviews are as follows:

Positive Reviews	572.041
Negative Reviews	98.555

The distribution of the genres are as follows:

RPG	128.627
Coop	122.462
Strategy	109.394
Platformer	108.930
FPS	107.598
Fighter	93.585

The reviews are balanced in terms of genre, but not in terms of sentiment. In order to provide a proper comparison, the machine learning will be trained on the whole dataset as well as on a down-sampled dataset, where the amount of positive reviews will match the amount of negative reviews. In the final analysis, the results of the down-sampled dataset will be used, but this thesis will include results from both datasets for additional data and transparency.

There are also reviews from the MOBA (multiplayer online battle arena) genre in the dataset, but since there was a very limited amount of games under that category, it was not possible to scrape enough reviews thus leading to the MOBA genre being omitted.

3 Sentiment Analysis: Packages and Algorithms

This section will detail the packages used as well as the chosen algorithms.

3.1 Scikit-learn

Most of the algorithms used in this paper are provided by the scikit-learn project[13]. Here is a brief overview of the algorithms and why they were chosen:

3.1.1 Naive Bayes

Naive Bayes was chosen for its simplicity as well as not being as efficient when it comes to feature dependency. This is especially interesting, because it offers an insight if steam reviews on average are written more plainly or complicated.

Two types of Naive Bayes algorithms were chosen, the first one being the multinomial type. It simply evaluates each word in the review and assigns it a certain probability of that word belonging to either sentiment class. Which leads to a review containing the words "good" or "best" have a high probability of belonging to the recommended sentiment class[20].

Furthermore, the key assumption for this algorithm is that the features are conditionally independent. This reduces the number of parameters that need to be estimated from the training data[20].

This allows for a train time complexity of O(n*m), where n is the number of samples and m is the number of dimensions or features. The prediction time complexity is O(c*m) where c refers to the class. For each class the feature has to be retrieved in order to make a prediction[20].

Fleizach [14], had limited success with an increasing amount of samples, as it decreased accuracy, but increased recall.

The second type of chosen algorithm is the Complement Naive Bayes. This algorithm is proficient in dealing with imbalanced datasets. This type of Naive Bayes assigns the weights by looking at all other classes combined except the class it is currently considering. In other words, it looks at a review and considers the adjusted weights by extracting information by looking at reviews exclusively from other classes. When talking about sentiment, this would mean when the algorithm observes a review classified as "positive", it will only consider reviews classified as "negative" to assign weights to adjust for the review it is currently considering, the one being of sentiment "positive" [1].

The dataset for this thesis features a significant imbalance in terms of sentiment (before the down-sampling), thus making CNB a solid choice.[3]

3.1.2 Logistic Regression

Logistic Regression will be explained on the basis of [25].

Logistic Regression is a statistical method to analyze data and predict a binary outcome. The paper [25] claims that logistic regression is better suited for predicting binary out-

[3]SKLearn Naive Bayes: https://scikit-learn.org/stable/modules/naive_bayes.html

comes than linear regression because linear regression does not take into account that the outcome variable is limited to two values. Odd ratios are a common way to express the results of logistic regression. The odd ratio explains how a one unit increase in the predictor variable influences the probability of a certain outcome variable. Logistic Regression can also be used to adjust for confounding variables, which ought to improve the predictions.

Logistic Regression was chosen, because it is made for binary tasks. The train time complexity of logistic regression equals to O(n*m), where n is the number of samples and m is the number of features. The prediction time complexity is O(m). This results in linear scaling.[4]

3.1.3 K-nearest neighbours

As described in [37], kNN is a simple yet effective algorithm.

kNN is non-parametric as well as supervised and is used for both classification and regression. The basic premise of this algorithm is that it matches a new unlabeled point with n closest data points, thus generating its prediction. Basically the majority of a certain class k in the number of neighbors n decide which class k the new unlabeled point is being assigned to. The steps include:

- Load the training data and the unlabeled data point.
- Calculate the distance between the new data point and each training example.
- Sort the training examples in ascending order, based on the calculated distance.
- Select the k nearest neighbors from that list.
- Assign the label of the majority of the k nearest neighbors to the new data point.

The training time complexity is O(k*n*m) where n is the number of samples, m the number of features and k the number of neighbors. The prediction time complexity is O(n*m). [5]

3.1.4 Random Forest

The basis for the description of this algorithm stems from[24].

The algorithm works as follows:

- Randomly select a subset of the training data with replacement to create a bootstrap sample.
- A bootstrap sample is used to estimate the distribution of a sample. This is done by repeatedly sampling with replacement from the original dataset[9].
- Construct a tree for each sample of the bootstrap. This means each tree stems from a different bootstrap sample.

[4]SKLearn LogReg: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[5]SKLearn kNN: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

- Randomly select a subset of features to consider for each tree.
- Combine the predictions from each tree in order to decide on the outcome. Majority is used for classification, while for regression averaging is used.
- With the combined predictions, a final decision is made.

The key advantages of this method are robustness to overfitting as well as being able to handle high dimensional data.

Lastly, the train time efficiency of this algorithm is O(k * m * log(n) * n), where as k is the number of trees, m is the number of features and n is the number of samples. The prediction time efficiency is O(m*k).[6]

3.1.5 Passive Aggressive Classifier

Described by Crammer et al [10], this algorithm follows two rules: If the prediction is correct, the model tends to incorporate small to no updates, thus being the passive side. If the prediction is incorrect on the other hand, the aggressive side takes over and procures large updates to the model. This entails that this model is only receiving major updates when a prediction fails to be correct.

In a more technical approach: Everything is centralized using weight vectors. Basically every new classification tries to find a new weight vector, which correctly classifies the current example, while staying as close as possible to the previous weight vector. This is also called a *constrained optimization problem*, where the objective is to minimize the distance between the new and old weight vectors. Minimizing the distance tends to be contradicting, since this is what the model wants to do: Incorporate large updates when being wrong. It is important to keep in mind, that the distance affects several aspects of the model: The larger the distance, the more the model forgets what it learned in the previous steps, which is not always beneficial. Keeping the distance close also means that there is a pattern, which may lead to good predictive power. All of this contributes to the constrained optimization problem where the objective is to find a good middle ground.

The runtime for this algorithm is O(n * m), where m is number of features and n is the number of samples. This is because the update step only requires a dot product and a scalar multiplication (because the algorithm deals with vectors) for each sample. [7]

3.1.6 Multi-Layer Perceptron (Neural Network)

[4] and [29] have been used to look up aspects of this algorithm and form the basis for the explanations below. Additionally, this [26] was used to broaden the understanding of this type of algorithm. Although the latter paper focuses on CNNs, it also laid the foundation for MLPs. It had a significant influence in pioneering this type of algorithm.

A short overview of the algorithm:

[6]SKLearn Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[7]SKLearn Passive Aggressive Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html

- MLPs are a *feed-forward neural network* used for either classification or regression tasks. They consist of an input and output layer and an arbitrary amounts of hidden layers.
- A *neural network* is designed to mimic human pattern finding. It consists of several nodes connected with each other, exchanging information and assigning weights to each of the nodes.
- *Feed-forward* describes the flow of information. It only flows in one direction. In the case of MLPs, the information flows from the input, through the hidden, into the output layer.
- Performance is significantly impacted by the amount of nodes in each hidden layer and the amount of hidden layers employed.
- The hidden layers assign weights which in turn influence the outcome. More does not always mean better, as too many nodes may be prone to overfitting as well as being computationally infeasible.
- By using the approach suggested in the first paper, the authors managed to achieve a higher classification accuracy in the classic iris dataset compared to other neural network models.
- The first paper managed to optimize the model while also lowering the amount of connections needed.

The runtime was this algorithm has several variables:

- n: Number of samples
- m: Number of features
- L: Number of layers
- h: Numbers of neurons in each layer
- k: Number of epochs (iterations)

For simplicity reasons, the following equations assume that the number of neurons in each hidden layer are the same.

First, the training time complexity: The forward pass through one layer h takes O(m * h) for the first layer and then for every subsequent layer O(h*h). This is the feed forward process. The backward pass has a similar runtime, since it follows the same path as the forward pass, just in reverse. The back pass part would then simply be added to the equation below, but since the big O notation drops constants, it is not part of the equation. This results in a runtime for one epoch of O(n*(m*h+(L-1)*h*h). The (L-1) is used for the subsequent layers, because the first layer is trained in the m*h sub-process, leaving L-1 layers to be trained on h*h runtime. The n represents the samples, because the calculation has to be done for each sample. Lastly, the training may also take several epochs. This results in the final equation of:

$$O(k * (n * (m * h + (L - 1) * h^2)))$$

Secondly, the prediction time complexity: The prediction does not back-propagate, because the weights do not need to be adjusted during the prediction. This results in a

prediction time complexity of

$$O(n(m * h + (L - 1) * h^2))$$

The variable n in this case refers to the number of samples to be predicted.

An in-depth explanation about neural networks, especially back propagation and gradient loss is given in section 3.2.1.

Nevertheless, neural networks allow models to learn about complex relationships. [8]

3.1.7 Gradient Boosting Classifier

This survey [16] looks at many types of different gradient boosting algorithms and serves as the foundation of understanding this type of algorithm. In order to properly comprehend this type of algorithm, [15] was chosen. The descriptions of the components are largely based on the aforementioned reference.

GBA has 3 key components:

- *Loss Function:* This is also used in the neural networks in tensor which is the primary objective of these algorithms. It measures how well the model predicts the target variable. It is quickly noticeable if the model is overfitting from this variable. The most common loss functions are usually MSE or Cross-Entropy[15].
- The weak learners: A simple model that predicts outcomes. Typically decision trees are being used as weak learners. Each of these weak learners main objective is to minimize the loss function[15].
- The core of the GBA is the additive model. It combines the predictions of multiple weak learners to create a strong predictive model. This means that in a step by step manner, components are being gradually added in order to construct the model. As it is additive, each weak learner is trained to correct the errors of its predecessor[15].

The problem when facing this algorithm in this paper was the lack of computational resources, resulting in the kernel constantly dying when trying to train on the original dataset. This is why a smaller subset had to be brought into play, in order to get any result out of this algorithm. More details will be discussed in the problems section.

The runtime for this algorithm is as follows: The tree construction takes O(n * m log n), where n is the number of samples and m the number of features. Fitting each tree takes O(d * n log n) where d is the depth (of the tree). Depending on the boosting stages denoted by T, the training time complexity takes the form of

$$O(T * (n * m * log(n) + d * n * log(n)))$$

The prediction time complexity is O(T*d). This is because each new sample is being applied to each tree in order to generate a prediction. For n samples, the prediction time complexity is:

$$O(n * (T * d))$$

[9]

[8] SKLearn MLP: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[9] SKLearn GBA: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

3.2 Tensorflow: Neural Networks

This section will briefly talk about the 2 types of neural networks used with the Tensorflow library. Tensorflow was chosen for its intuitiveness, ability to utilize a GPU as well as their thorough documentation[12].

3.2.1 Recurrent Neural Network

These types of RNN usually use a Long Short-term Memory layer. The paper by Hochreiter [17] formed the foundation for the now widely adopted LSTM architecture, which is especially effective for learning long-term dependencies by introducing special memory cells in order to counteract the vanishing gradient problem.

In order to gain a full understanding, a brief explanation of a few concepts will follow. The basis for the explanations originate from the paper mentioned above and [30]. The paper "Learning representations by back-propagating errors" by Rumelhart et al is an influential paper, forming a part of the foundation for deep learning.

- *Back propagation*: In order for neural networks to learn from their mistakes, the algorithm takes errors from the output and directs it back towards the input layer (basically the reverse direction of the feed forward neural network). This is done in order to calculate the loss value of the corresponding loss function (see gradient boost classifier). This information is then used to update the weights of each node.
- *Gradient of the loss function*: It tells how much the loss value is changed when small alterations are made to the parameters. This is the core aspect of these types of machine learning algorithms. It is used to optimize the model.
- *Gradient descent*: A common optimization algorithm. It uses the gradient to create an optimal model by using iteration to minimize the loss function.
- *Unrolling*: This process is responsible for remembering. It is like reading a story, each word is being read and used to update the memory of the algorithm. The more it reads, the more it tries to find patterns. The story in this case is the data. Now the longer the story, the more the RNN may tend to forget what happened at the beginning. This leads to the next discussed point:
- *The vanishing gradient problem*: As gradients travel back during the back propagation phase, they become smaller. The problem describes how these gradients fail to properly update layers which are nested deep within the neural network (or at the very start). This is preventing effective learning, because the nodes deep within the layer cannot properly adjust their weights, because the gradients become too small.

So a neural network training instance consists of 2 stages: The forward stage when predictions are generated and a backward stage, which is responsible for adjusting the weights.

A LSTM unit consists of following elements:

- Cell: Stores information over time intervals
- Input gate: This regulates the flow of new information into the cell. The gate decides which information to store (or to forget).

- Output gate: This regulates which information is to be passed on, based on the current state of the cell. It also uses the previous states to generate its decisions.
- Forget gate: This regulates which information is to be discarded from the previous states.

The runtime is dependent on the size of batch, buffer and dataset as well as the chosen parameters for each layer in the neural network model. A general calculation for the runtime would be:

LSTM Layer: Based on the discussed gates above, the runtime can be generally defined like this:

- Input Transformation: O(m*h), where m represents the features and h the amount of hidden units. The input has to be transformed to a suitable format, which is given by the hidden layer.
- Hidden State Transformation: O(h*h). Here the hidden layers interact with each other. The hidden layer is being multiplied with the hidden layer of the previous time step.
- Gate calculations: Each gate needs to transform the input data to a valid format. That format as mentioned above is given by the hidden layer. The gates all work independently, which means that the input and hidden state transformation is happening for each gate.

With these points in mind, the training time complexity per time step can be generally denoted as:

$$O(4 * h * (m + h))$$

The number 4 stems from the 3 gates and the cell state. In order to account for the sample size, which has been batched in this thesis, the finalized form of the time complexity equation would be:

$$O(k * (b * t(4 * h * (m + h))$$

With k being the number of epochs, b the batch size and t the sequence length. The runtime definition given takes a lot of generalization into account. The runtime, as mentioned above, is dependent on a lot of factors not excplicitly mentioned in the O time.

The core strength of this algorithm is its ability to remember patterns for a long "time", which is especially beneficial for complex data, where patterns are wide reaching. On the other hand LSTM is prone to overfitting if poorly implemented.

Overall this can be considered a milestone for machine learning, thanks to its ability to remember long term dependencies in data.

Another well known architecture which is tackling the problem mentioned above are Gated Recurrent Units. Cho presents in his paper [11] alternatives to the LSTM layer. GRU are simpler and yet perform nearly as well as the critically acclaimed LSTM layer.

In summary, GRUs manage to prune down the necessary gates to 2 instead of the 3 mentioned above for the LSTM layer. These gates are called the update and reset gate.

Using the same rules as above to calculate the runtime efficiency, the equation takes this form:

$$O(k * (b * t(3 * h * (m + h))$$

Another paper claiming the superiority of GRUs in terms of event classification in sports videos is "An Experimental Comparison of Deep LSTM and GRUs for Event Classification" by Kajal Chandani[8].

In that paper, GRUs generally performed better in terms of runtime, accuracy and precision compared to the LSTM layer. In addition it used less training parameters, adding to the efficiency.

Tensor offers both layers. [10]

3.2.2 Convolutional Neural Network

To explain the foundational structure of CNNs, a paper by O'shea et al [28] is going to be used.

CNNs operate within 3 layers:

- *Convolutional Layer*: This layer is responsible for generating a feature map based on filtering a small region of the input data.
- *Pooling Layer*: These generated feature maps are then passed through pooling layers, which are responsible for reducing the spatial dimensions of the data.
- *Fully connected Layers*: The output of the previous layer is than passed through the fully connected layers, which map the extracted features into the final output.

During training, *back propagation* and *gradient descent* is used to minimize the difference between prediction and true labels.

CNN showed exceptional proficiency at identifying local patterns, such as contrasting borders in an image, but it has also proven its merit when processing textual data as shown in this paper by Yoon Kim [22]. Thanks to this algorithms ability to identify local patterns, it is usually used for image classification[23].

In order to determine the runtime, each layer has to be analyzed:

- Embedding: O(n*(L*d)), where n is the number of samples, L is the length of each text (can also be seen as word count or tokens) and d being the features of the embedding. Basically the length of each text is being embedded into a vector of dimension d.
- Convolution: O(k * (H * W * f * d)), where k is the amount of filters, H the output height after convolution, W the output width after convolution and f the filter size (how many words for each filter). Basically the algorithm takes an f-sized window and puts it over the embedded representation of the text. This is done for each filter.
- Fully Connected Layers: This is dependent on the neurons in each layer and is difficult to generalize.

[10] Tensor RNN: https://www.tensorflow.org/guide/keras/working_with_rnns

The final form of the training time complexity is:

$$O(n * (k * (H * W * f * d))$$

The embedding is omitted because it would have been added to the equation and it is usually not as intensive as the actual convolution. Following the big O form rules, this leads to omitting that part of the run time efficiency equation. Nevertheless, the O form given generalizes a lot and should only be a rough indication of the runtime.[11]

[11] Tensor CNN: https://www.tensorflow.org/tutorials/images/cnn

4 Genre Classification: Packages and Algorithms

This section will talk about the packages used as well as the chosen algorithms.

4.1 Scikit-learn

As stated in the previous section, most algorithms, even for the genre classification, are implemented in the scikit-learn library[13].

4.1.1 Decision Tree

The decision tree algorithm implemented in sci kit is based on the CART algorithm. This specific version is explained in this paper by Jiang Su et al [36]. A more general explanation can be found in [27] and in [7].

A brief summary on how it works:

- *CART Binary Tree*: It builds a binary tree by recursively partitioning the data based on a single feature. This tree is grown by finding the feature and threshold that best separates these classes. Each tree is supposed to create optimal separation so that the algorithm can make accurate predictions.
- A *feature* in this sense is a numerical representation of a word. This means a review would be split recursively into single words in order to decide where the best split would be. For example, if the word good is present in the review, the model may decide that this feature alone is enough to predict the review being positive. On the other hand, it may decide that good is not enough of a concrete predictor and decide for further splits and so on. These splits create the binary tree.
- *Gini Impurity*: Describes the likelihood of an incorrect classification based on a new instance of data if that prediction was generated randomly. The randomness is being decided upon the distribution of class labels from the dataset.
- *Classification*: CART uses the Gini impurity as the splitting criterion. The trees, as explained above, are created on the rule that the Gini impurity is minimized. The lower the Gini value, the purer the split.
- The tree growth continues until a *stopping criterion* is met. This can be for example a minimum number of leafs for each node. In order to avoid overfitting, these trees are then pruned back using methods like cost-complexity pruning.

New data points are then being fed into these trees in order to generate a prediction. The time complexity for training depends on the number of samples n, the number of features d and the maximum depth of the tree m. The runtime will be defined on the basis of the worst case scenario runtime wise.

There are 2 parts of the decision tree building:

1. Splitting: O(n*d*d).
2. Building the tree: O(n * d * d *m).

This leads to a training runtime efficiency of

$$O(n * d^2 * m)$$

The splitting part is being dominated by building the tree, which leads to the O time equation being equal to the O notation of the tree building part.

Predictions on the other hand follow a O(t) runtime efficiency, where t is the depth of the tree.

Overall, this algorithm is easy to interpret.[12]

4.1.2 K-nearest neighbours

See section 3.1.3.[13]

4.1.3 Logistic Regression

See section 3.1.2. As mentioned on the sci kit site, logistic regression can handle multi-classing as well. Another example for its effectiveness for these types of tasks can be found in the paper "Efficient methods for online multiclass logistic regression" by Agarwal et al [2]. [14]

4.1.4 Random Forest

See section 3.1.4. A paper by IJSRCEIT [3] showcases random forests being successfully utilized in a healthcare setting. The RF classifier outperformed other algorithms in all usual score metrics. It is also one of the few papers focusing on multiclassing with random forests. [15]

4.1.5 Bernoulli Naive Bayes

The explanation of this algorithm is based on this paper by Ismail et al [19].

How this works:

- The core concept is that the algorithm looks at all features and assumes each one to be binary.
- It assumes each feature is independent from one another.

It looks at each feature, assigns it a boolean value and predicts upon a feature being present or not. The training time complexity is:

$$O(n * d + c * d)$$

where n is the number of samples, d the number of features and c the number of classes. Since there are 2 terms being added together, the dominant part of these 2 terms decides the runtime efficiency. For this thesis specifically, where the dataset is large and rich in dimensions, the (n*d) part is dominant, resulting in a shorter O form of O(n*d).

[12]SKLearn Decision Tree: https://scikit-learn.org/stable/modules/tree.html

[13]SKLearn kNN: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[14]SKLearn Logistic Regression: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[15]SKLearn Random Forest: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

The prediction time complexity is:

$$O(d + c * d)$$

The equation can also be shortened by looking at the dominating term, which is (c*d), because the variable d is present in both terms.

This makes this algorithm efficient, especially when the vocabulary size is large.

Since there will be two versions of the machine learning training, one with genre tags included in the dataset and one without, the aforementioned rules of this algorithm may have a significant impact on the performance. [16]

4.1.6 Quadratic Discriminant Analysis

The description that follows is based on the official paper that forms the foundation for the sci kit implementation by Srivastava et al [34].

The key characteristics are:

- *Bayesian estimate*: In order to estimate the parameters of the Gaussian distribution for each class, Naive Bayes is used.
- QDA assumes that each class follows a *Gaussian distribution.*
- For each class, QDA requires a *covariance matrix.* This is computationally expensive and can be a problem for large datasets.
- QDA solves a *quadratic equation* to determine the decision boundary, which once again, is computationally expensive.

The training runtime is based on 3 parts:

- Prior Probability: O(c). For each class (c) the prior probability is being estimated.
- Means of each Class: O(n*d), where n is the number of samples and the number of features.
- Covariance Matrix: O(c*d*d). The assumptions is made that each covariance matrix has a dimension of d * d.

This leads to a training runtime efficiency of:

$$O(c + n * d + c * d^2)$$

For the prediction, a discriminant function has to be calculated for each class. Afterwards, the highest discriminant value has to be found among all classes. This leads to a runtime of

$$O(c * d^2)$$

The reason why a subset has been used for the algorithm in this thesis, is because the server could not handle the large dataset when training using QDA. This will be explained in more detail in the Problems section. [17]

[16] SKLearn Naive Bayes: https://scikit-learn.org/stable/modules/naive_bayes.html

[17] https://scikit-learn.org/stable/modules/generated/sklearn.discriminant_analysis.QuadraticDiscriminantAnalysis.html

4.1.7 Gaussian Naive Bayes

This paper explains the algorithm [21].

Gaussian Naive Bayes has the following attributes:

- Assumption that features are independent and follow a Gaussian distribution.
- Afterwards, GNB calculates the mean and standard deviation of each feature for each class. By using the Gaussian probability density function, it calculates the likelihood of a new instance belonging to each class.
- Afterwards, the Bayes theorem is applied to determine the probability of each class.

This algorithm is very similar to the QDA in terms of steps. It also calculates the means and prior probability of each class. The difference lies in the calculation of the third step: instead of a covariance matrix, the GNB calculates the class variance. This results in a training runtime efficiency of:

$$O(c + n * d + c * d)$$

The prediction time also benefits from this change: The prediction runtime efficiency is:

$$O(c * d)$$

Which makes this algorithm more efficient than the QDA.

There is also a paper about the GNB and the classic iris dataset [18], which led to the decision to include this algorithm.

The algorithm also needed a smaller subset to be used, otherwise the kernel would die.[18]

4.1.8 Multi-Layer Perceptron (Neural Network)

See section 3.1.6.[19]

4.2 Tensorflow: Neural Networks

The descriptions from the sentiment section also apply to the tensorflow algorithms for the genre classification.[12]

4.2.1 Recurrent Neural Networks

See section 3.2.1. [20]

4.2.2 Convolutional Neural Network

See section 3.2.2. [21]

[18]SKLearn GNB: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB
[19]SKLearn MLP: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[20]Tensorflow RNN: https://www.tensorflow.org/guide/keras/working_with_rnns
[21]Tensorflow CNN: https://www.tensorflow.org/tutorials/images/cnn

5 Preparation for the Machine Learning

Now that the data has been properly cleaned, it is time to split it into training and test data. There were two approaches where the data has been prepared, one for each library used: Scikit-learn and Tensorflow.

For Scikit-Learn, it was decided to split the data between 80% train data and 20% test data. Additionally, depending on the algorithm, it was necessary to generate a sparse and a dense version of the data. When numeric input was required, TfIdfVectorizer has been utilized.

For Tensor it was decided to use the built in tensor objects to maximize performance. Once again a 80 / 20 split was chosen for the data. After determining the buffer and batch size, the actual datasets were generated and were ready to be fed into the neural networks.

Furthermore, for certain algorithms a subset had to be prepared in order to not having to exclude the aforementioned algorithms. For the sentimental part, the subset contained 50k reviews for each sentiment, resulting in a dataset with 100.000 reviews. For the genre classification, the subset included 15.000 reviews per genre. Since there were 6 genres, it resulted in a dataset with 90.000 reviews.

If an algorithm allowed the input of weights, then this was done. The weights were automatically assigned to a dictionary and passed as a parameter to the viable algorithms with the intent to boost performance. The weights can be seen in the jupyter notebooks available on the github page.

The sentiment analysis will also include 2 sections, one with the unbalanced dataset and one with the down-sampled classes, resulting in an equal distribution of the sentiment.

Lastly, for the training of the models within the tensor environment, WU Vienna has allowed access to their Nvidia GPU A30, greatly accelerating the rate of learning. They also provided sufficient RAM and CPU capabilities to ensure this thesis runs along smoothly.

6 Evaluation: Sentiment Analysis (unbalanced Dataset)

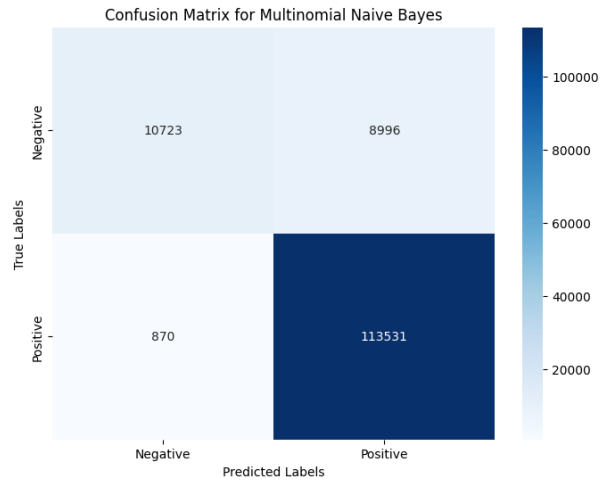
In order to evaluate the algorithms, certain performance parameters were chosen. Accuracy, recall and precision will be used in the sentimental part to portray the performance of each algorithm. Additionally, runtime, loss and subset used are going to be included where applicable. Furthermore, a confusion matrix [32] will be used to visually display the results. For the tensorflow algorithms, graphs will be used which showcase the accuracy and loss values for each epoch, further giving insight on the performance of these models.

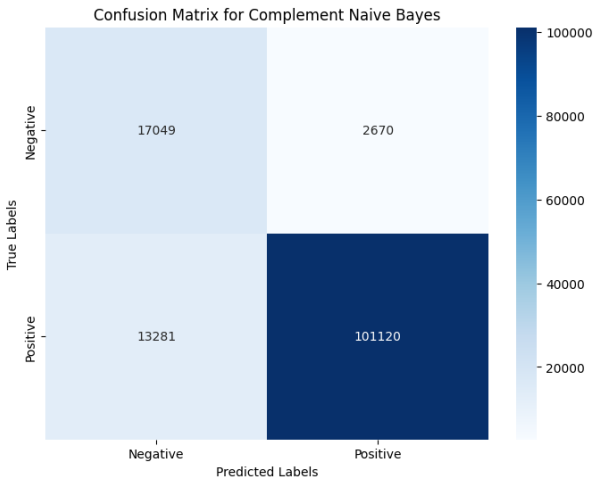
Runtime is measured with the inbuilt time command in jupyter. Accuracy is calculated with the inbuilt methods of both sci-kit and tensor. Loss is calculated in the tensor training session. For more details, please refer to the github repository and the sources within this paper and the notebooks.

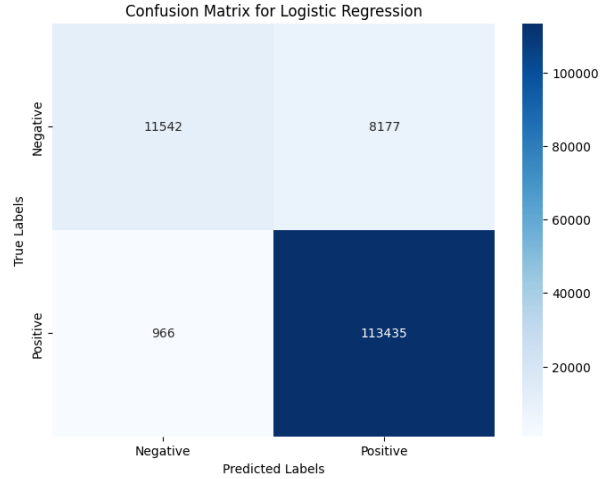
What follows is a series of Confusion Matrices, each labeled according to the algorithm

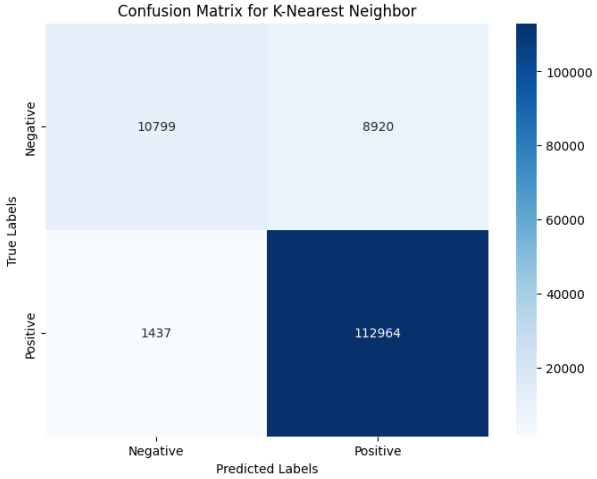
it represents. Afterwards, a table will be presented with different performance variables. Lastly, an analysis will close the sentiment analysis chapter.

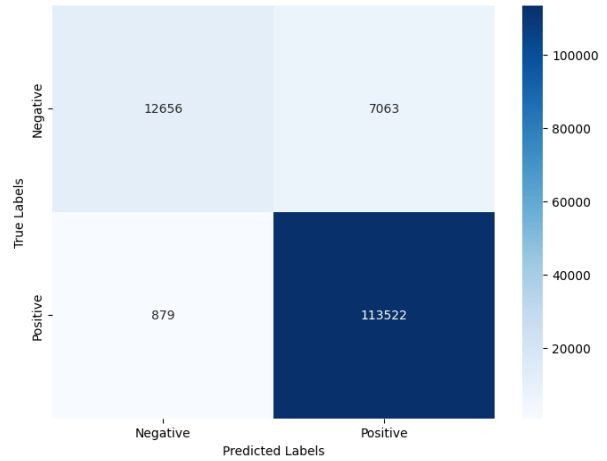
6.1 Confusion Matrices

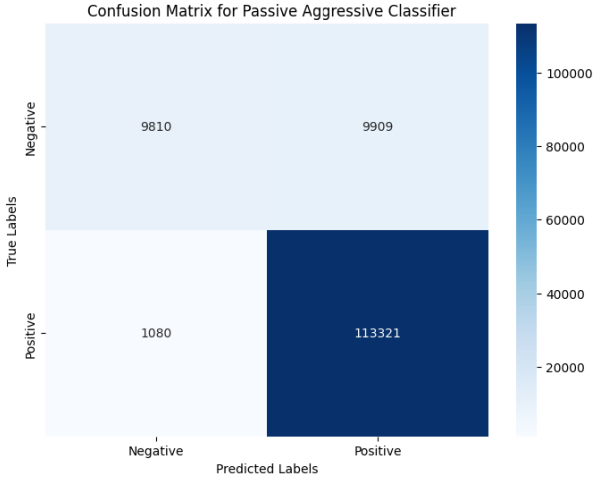

Confusion Matrix for Multinomial Naive Bayes
10723
8996
870
113531
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000

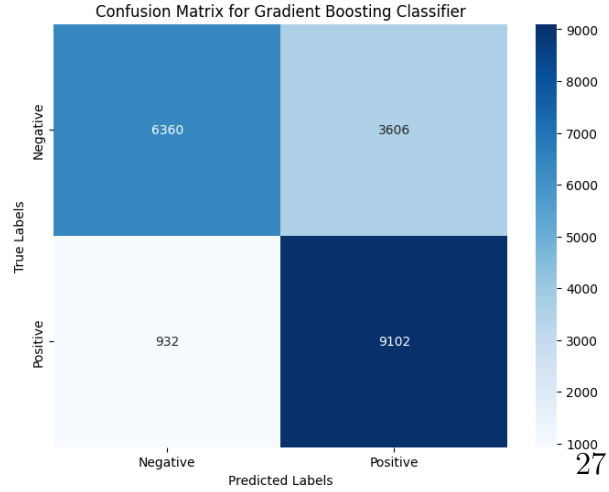

Confusion Matrix for Complement Naive Bayes
17049
2670
13281
101120
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000


Confusion Matrix for Logistic Regression
11542
8177
966
113435
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000


Confusion Matrix for K-Nearest Neighbor
10799
8920
1437
112964
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000

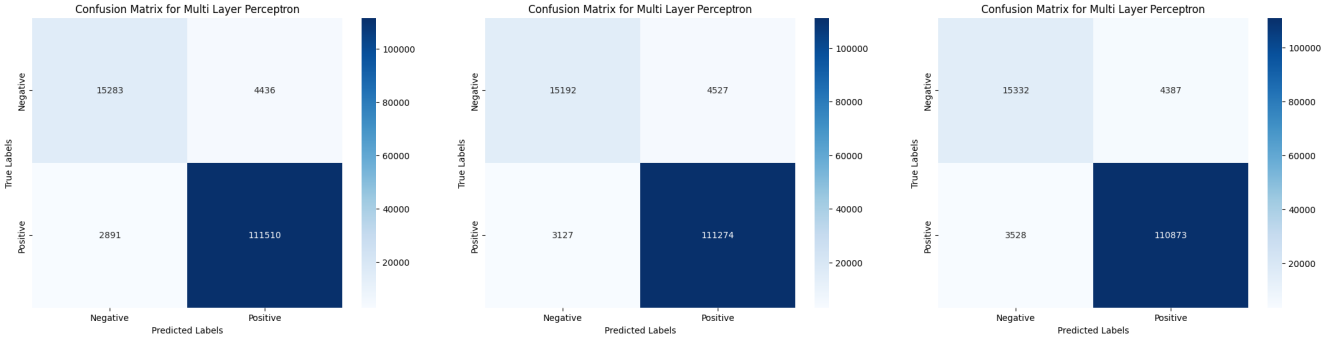

Confusion Matrix for Random Forest
12656
7063
879
113522
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000

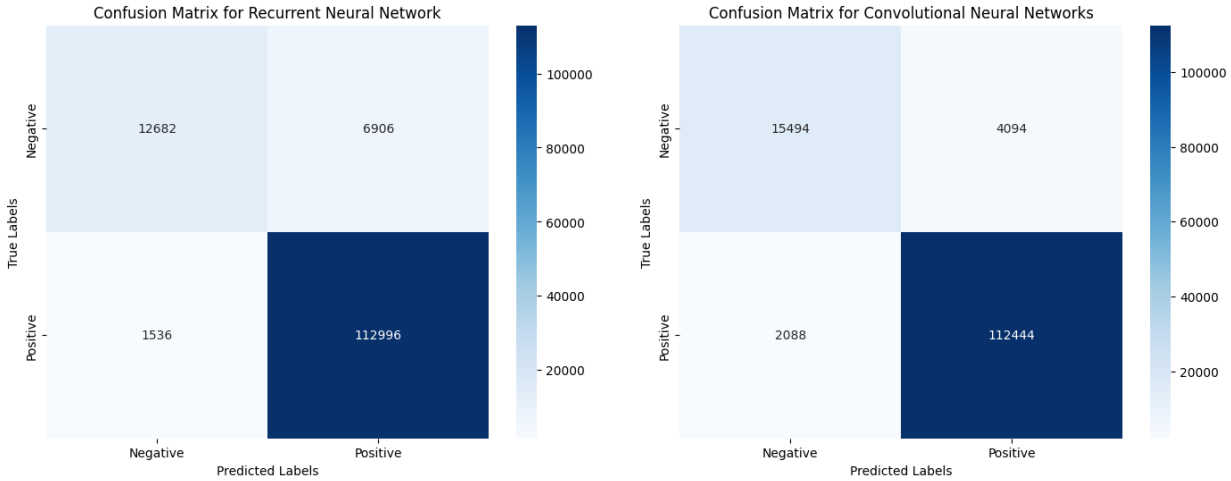

Confusion Matrix for Passive Aggressive Classifier
9810
9909
1080
113321
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000


Confusion Matrix for Gradient Boosting Classifier
6360
3606
932
9102
Negative
Positive
True Labels
Predicted Labels
9000
8000
7000
6000
5000
4000
3000
2000
1000

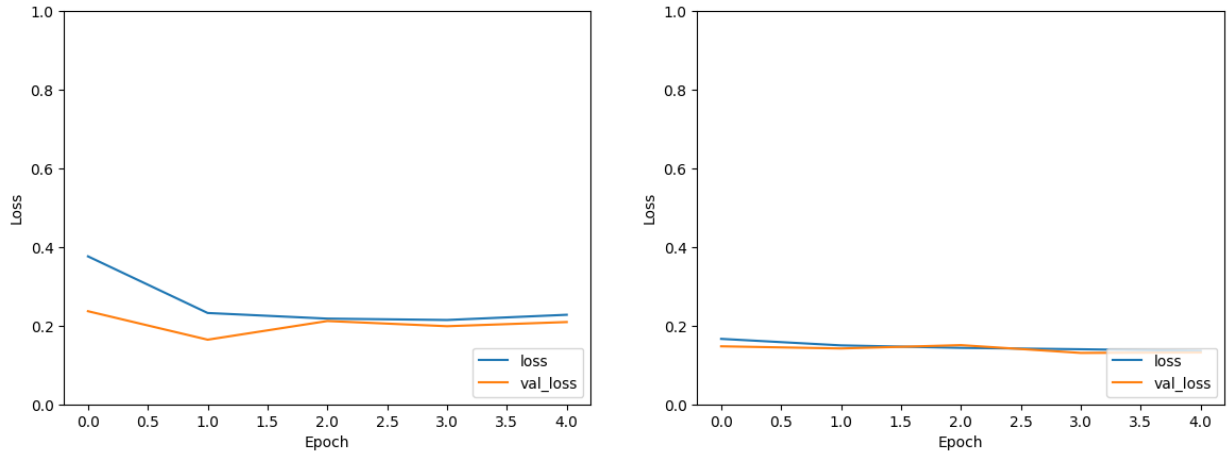
The following three figures had the following hidden layers assigned to them (from left to right):

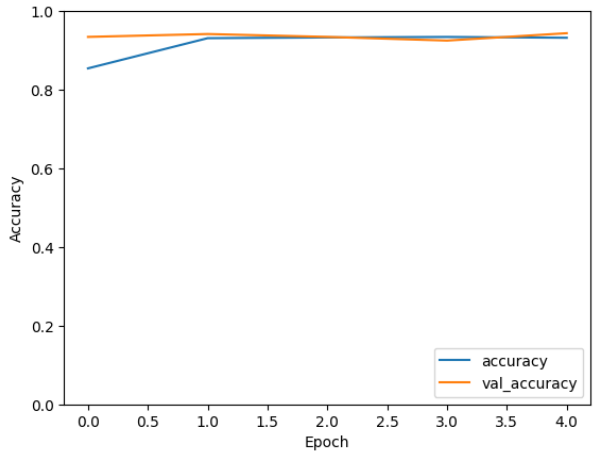
1. (100)
2. (100,50)
3. (200,100)

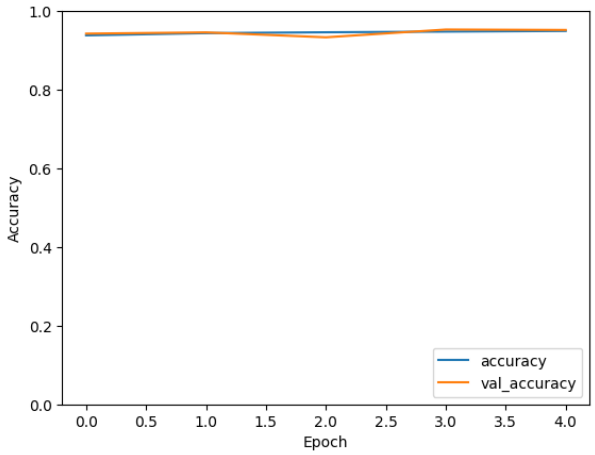




For these graphs, the left one always corresponds to the RNN and right one to the CNN.







6.2 Performance Table

Model	Accuracy	Precision	Recall	Runtime	Loss	Subset
Multinomial Naive Bayes	0.93	0.93	0.99	1.71s	NaN	No
Complement Naive Bayes	0.88	0.97	0.88	277ms	NaN	No
Logistic Regression	0.93	0.93	0.99	5min 19s	NaN	No
K Nearest Neighbor	0.92	0.93	0.99	20min 22s	NaN	No
Random Forest	0.94	0.94	0.99	1h 2min 48s	NaN	No
Passive Aggressive Classifier	0.92	0.92	0.99	8.39s	NaN	No
Gradient Boosting Classifier	0.77	0.72	0.91	49.8s	NaN	Yes
Multi Layer Perceptron (100)	0.95	0.96	0.97	1h 37min 28s	NaN	No
Multi Layer Perceptron (100,50)	0.94	0.96	0.97	1h 51min 14s	NaN	No
Multi Layer Perceptron (200,100)	0.94	0.96	0.97	8h 58min 17s	NaN	No
Convolutional Neural Network	0.95	0.96	0.98	9min 37s	0.14	No
Recurrent Neural Network	0.93	0.94	0.99	36min 52s	0.23	No

6.3 Analysis

Each model performed the task well. Every model besides Complement Naive Bayes and the Gradient boosting classifier has an accuracy above 90 percent. Recall has been near perfection with values around 97 to 99 percent. The poor performance of the gradient boosting classifier can be attributed to the subset and may have been significantly better if a larger dataset could have been used.

Runtimes range from mere milliseconds to nearly 9 hours. These times are not surprising, considering the mentioned runtime efficiency at each explanation of the algorithm.

In summary, every model tested has proven to be effective at the given task.

The good results may be due to overfitting and heavy unbalancing. For this reason, the final analysis will consider only the results of the following section.

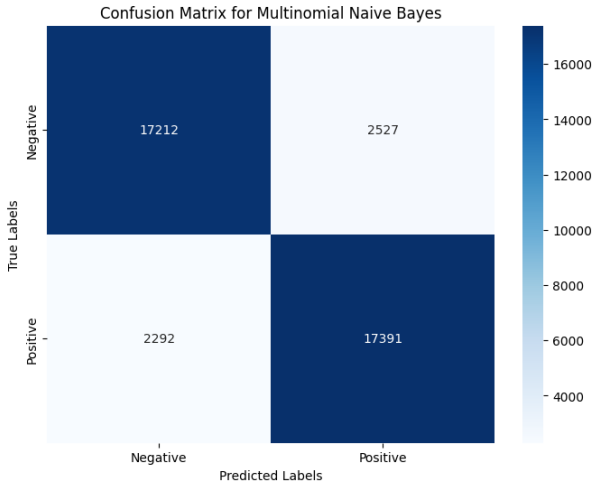
7 Evaluation: Sentiment Analysis (balanced Dataset)

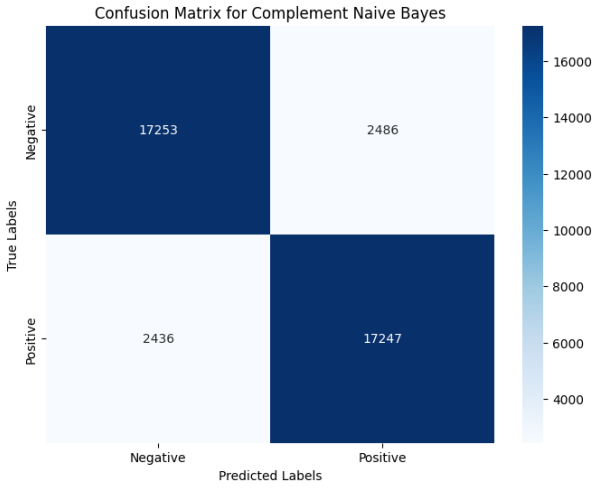
In order to balance the datasets, down-sampling has been used. The positive reviews have been sampled to match the negative reviews in numbers. This changes the count of Reviews for this section to:

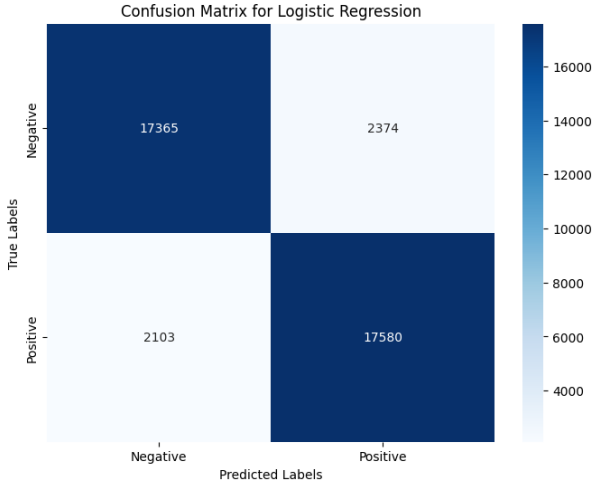
Positive Reviews	98.555
Negative Reviews	98.555

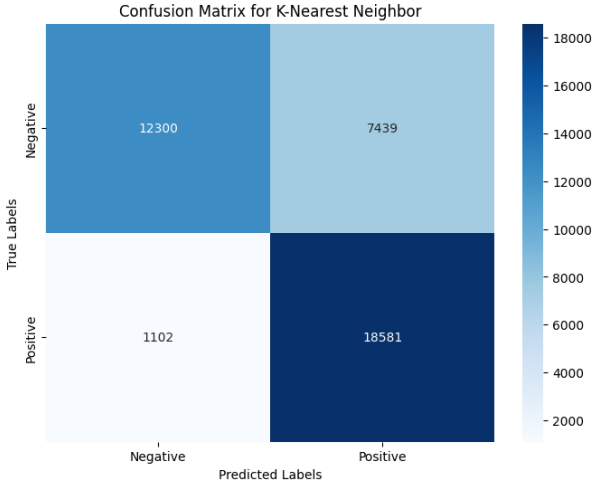
Everything will be identical to the previous section, besides the numbers and the analysis.

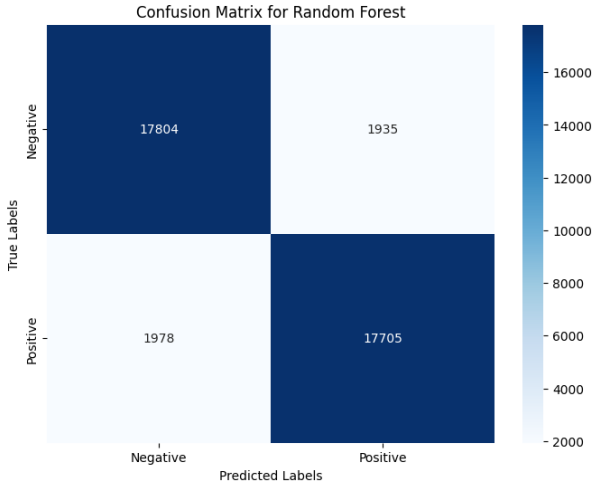
7.1 Confusion Matrices

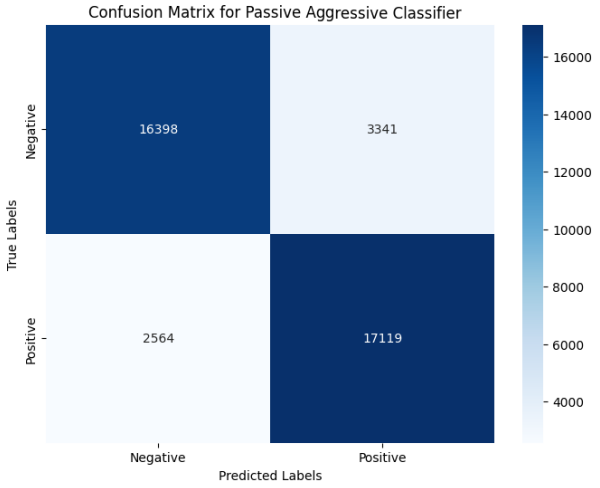

Confusion Matrix for Multinomial Naive Bayes
17212
2527
2292
17391
Negative
Positive
True Labels
Predicted Labels
16000
14000
12000
10000
8000
6000
4000

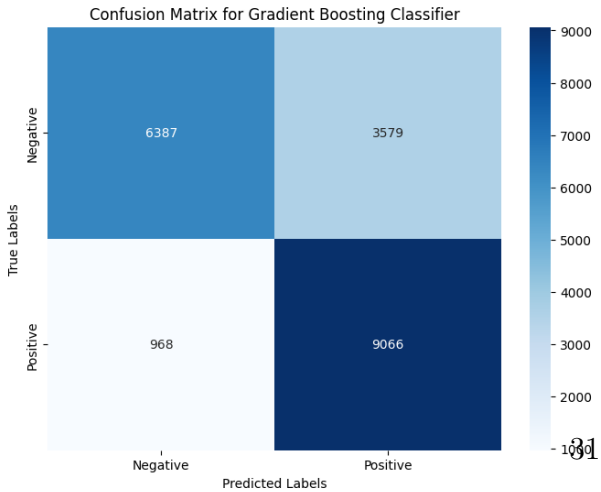

Confusion Matrix for Complement Naive Bayes
17253
2486
2436
17247
Negative
Positive
True Labels
Predicted Labels
16000
14000
12000
10000
8000
6000
4000


Confusion Matrix for Logistic Regression
17365
2374
2103
17580
Negative
Positive
True Labels
Predicted Labels
16000
14000
12000
10000
8000
6000
4000


Confusion Matrix for K-Nearest Neighbor
12300
7439
1102
18581
Negative
Positive
True Labels
Predicted Labels
18000
16000
14000
12000
10000
8000
6000
4000
2000

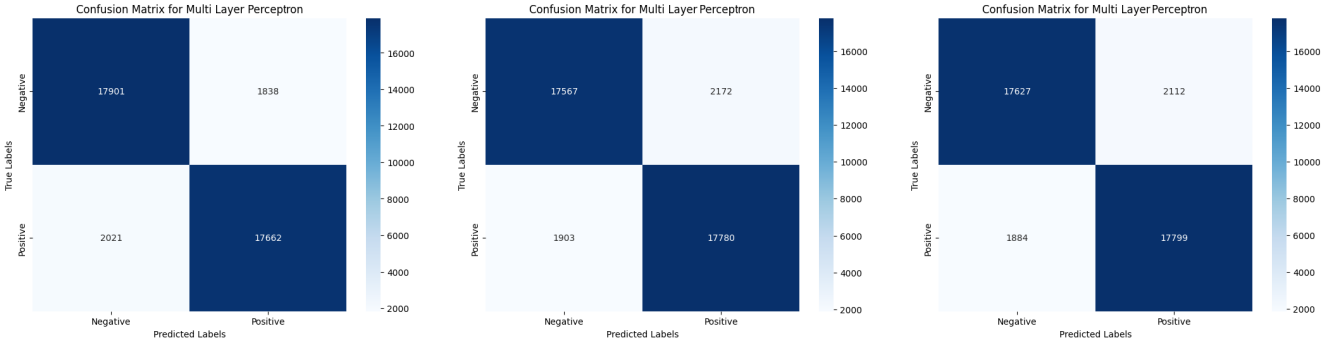

Confusion Matrix for Random Forest
17804
1935
1978
17705
Negative
Positive
True Labels
Predicted Labels
16000
14000
12000
10000
8000
6000
4000
2000

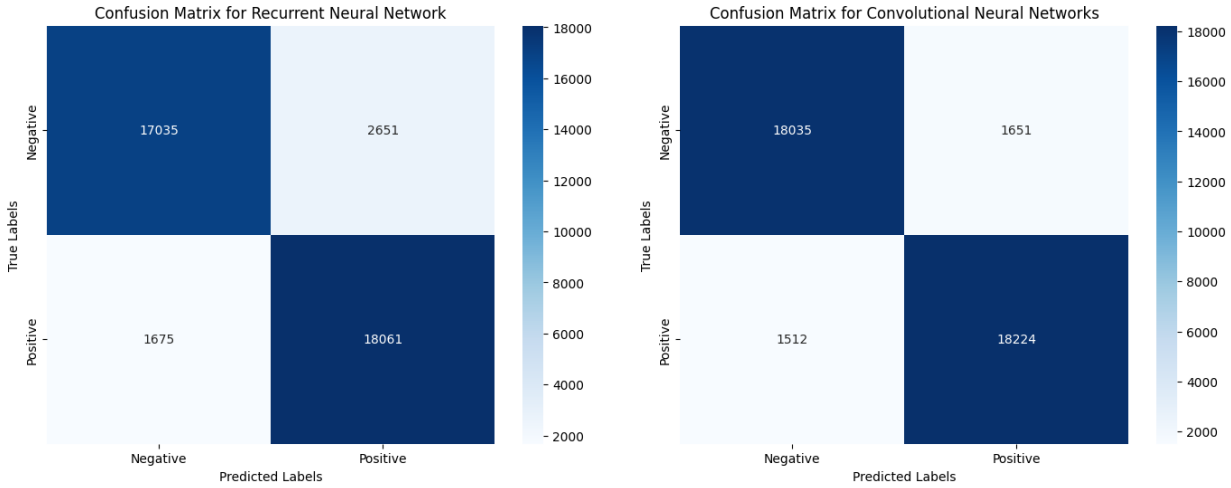

Confusion Matrix for Passive Aggressive Classifier
16398
3341
2564
17119
Negative
Positive
True Labels
Predicted Labels
16000
14000
12000
10000
8000
6000
4000


Confusion Matrix for Gradient Boosting Classifier
6387
3579
968
9066
Negative
Positive
True Labels
Predicted Labels
9000
8000
7000
6000
5000
4000
3000
2000
1000

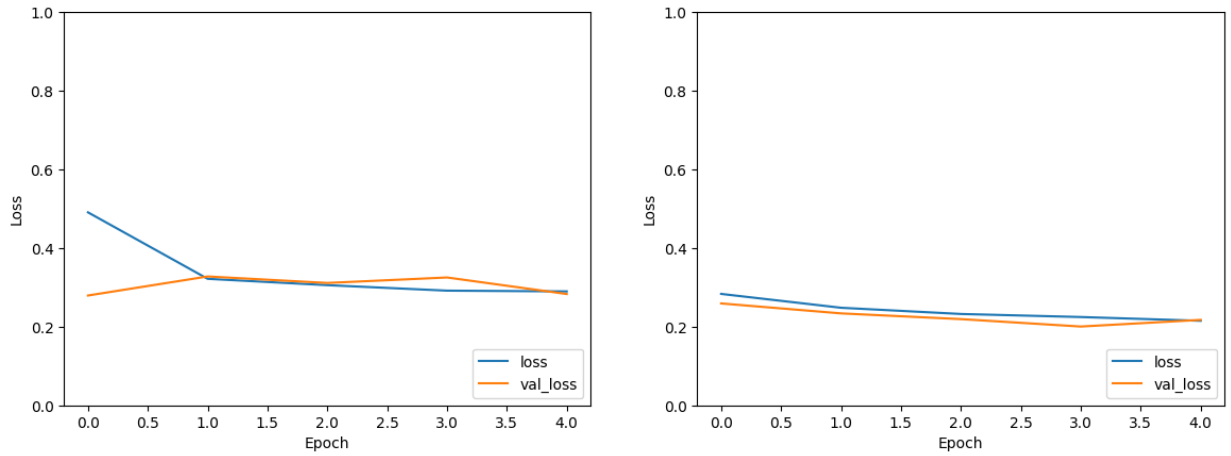
The following three figures had the following hidden layers assigned to them (from left to right):

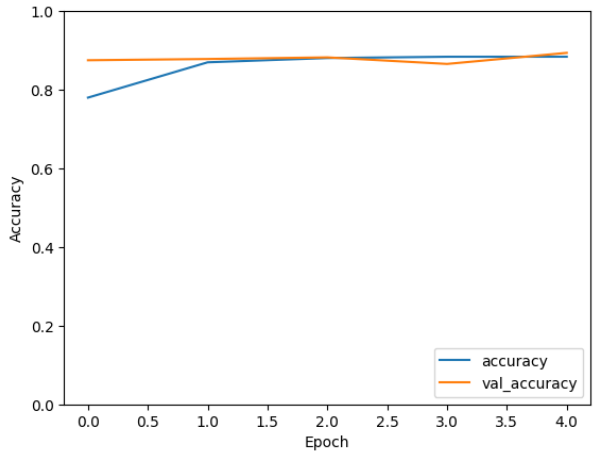
1. (100)
2. (100,50)
3. (200,100)

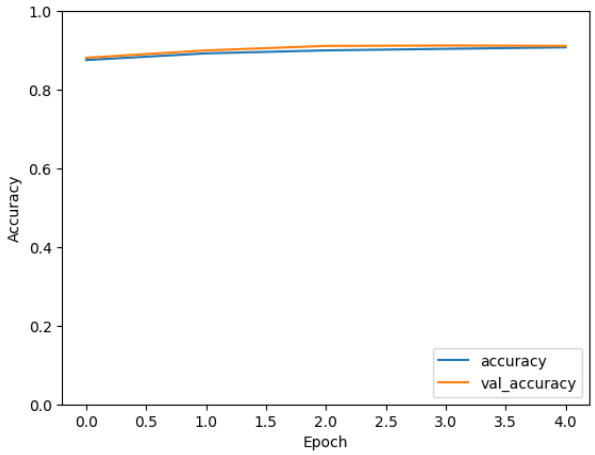




For these graphs, the left one always corresponds to the RNN and right one to the CNN.





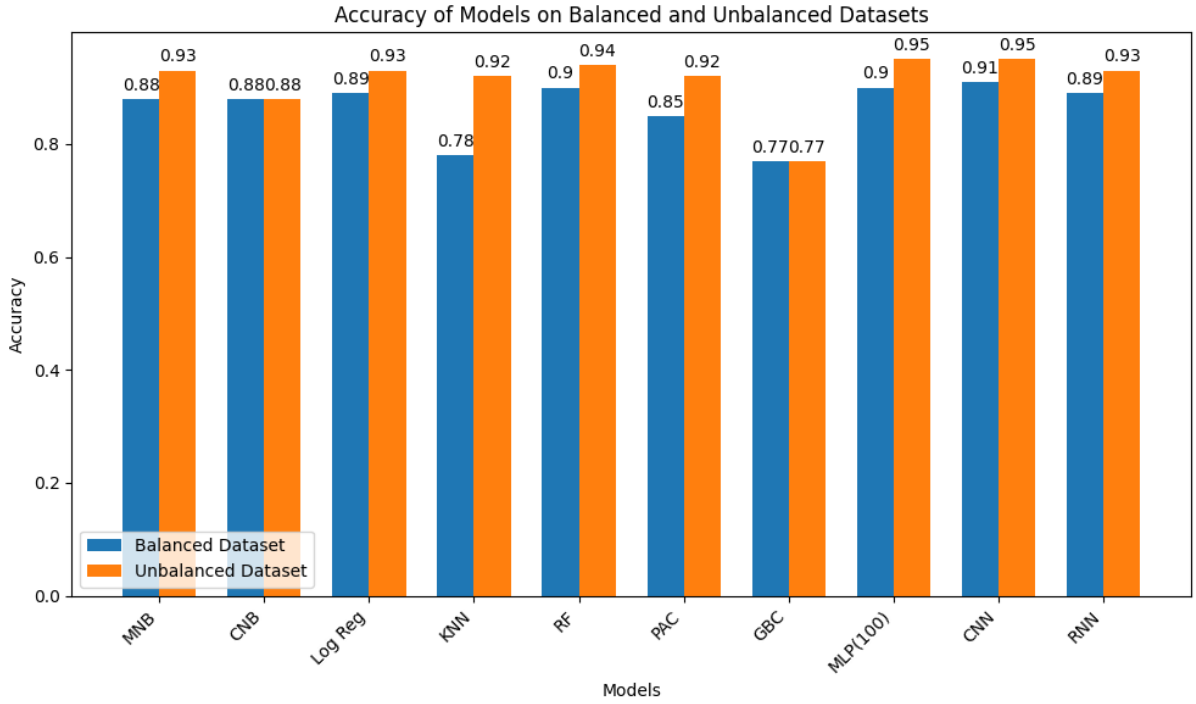


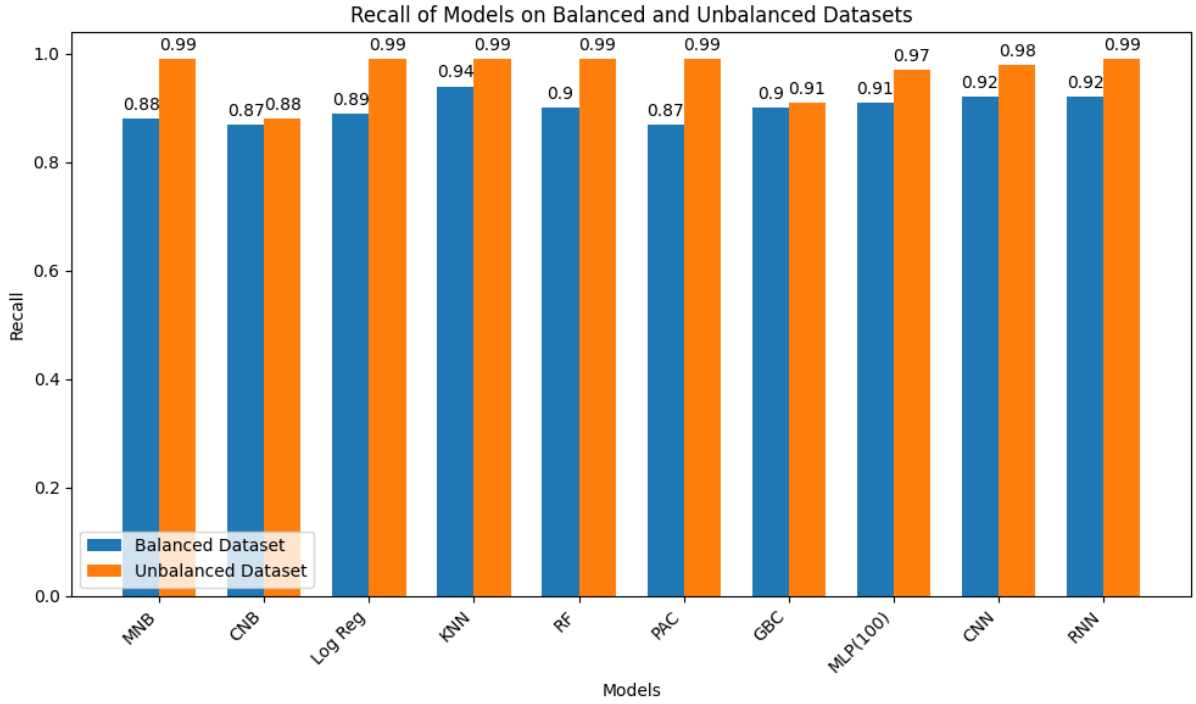
7.2 Performance Table

Model	Accuracy	Precision	Recall	Runtime	Loss	Subset
Multinomial Naive Bayes	0.88	0.87	0.88	256ms	NaN	No
Complement Naive Bayes	0.88	0.88	0.87	263ms	NaN	No
Logistic Regression	0.89	0.88	0.89	2min 23s	NaN	No
K Nearest Neighbor	0.78	0.71	0.94	3min 9s	NaN	No
Random Forest	0.90	0.90	0.90	11min 6s	NaN	No
Passive Aggressive Classifier	0.85	0.84	0.87	2.16s	NaN	No
Gradient Boosting Classifier	0.77	0.72	0.90	51.9s	NaN	Yes
Multi Layer Perceptron (100)	0.90	0.90	0.91	33min 30s	NaN	No
Multi Layer Perceptron (100,50)	0.90	0.89	0.90	25min 47s	NaN	No
Multi Layer Perceptron (200,100)	0.90	0.89	0.90	38min 52s	NaN	No
Convolutional Neural Network	0.91	0.92	0.92	4min 2s	0.21	No
Recurrent Neural Network	0.89	0.87	0.92	13min 43s	0.29	No

7.3 Analysis

The values are now worse than in the unbalanced sentiment analysis section. This might be because of the heavy unbalance causing the models to prefer one sentiment over the other. This means that if a dataset has 90% positive reviews, the model learns to only predict the sentiment "positive", because it will be right 90% of the time. This is apparent when analyzing the sentiment distribution of the original dataset. Although the models in the unbalanced section did not exclusively predict one sentiment, it could have still developed a strong bias towards the dominating class. The following two graphs compare the accuracy and recall of both sentiment analyses[31]:





As one can see from the graphs, the recall metric is always worse with the balanced dataset, for some models even 10 percent less. Accuracy is on average (excluding the models which used a subset) 5.2 percent worse with the balanced dataset. kNN performed especially bad with just 78 accuracy with the balanced dataset and 92 percent accuracy in the unbalanced dataset.

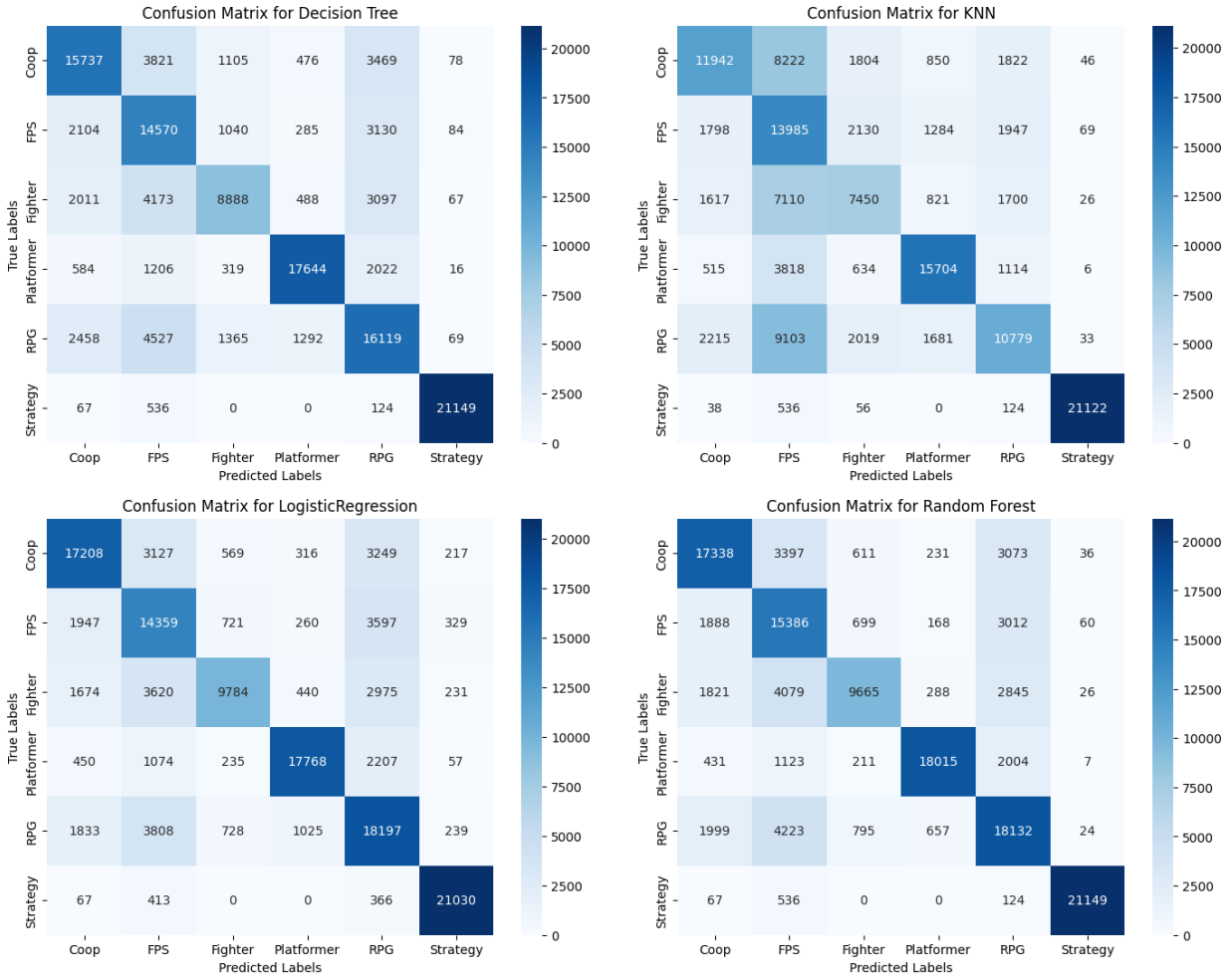
Nevertheless, the models do a good job of predicting the sentiment, with most models still converging towards the 90 percent mark.

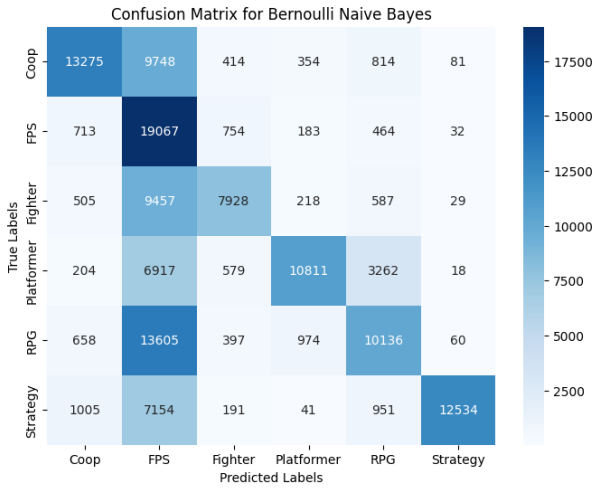
8 Evaluation: Genre Classification with Keywords

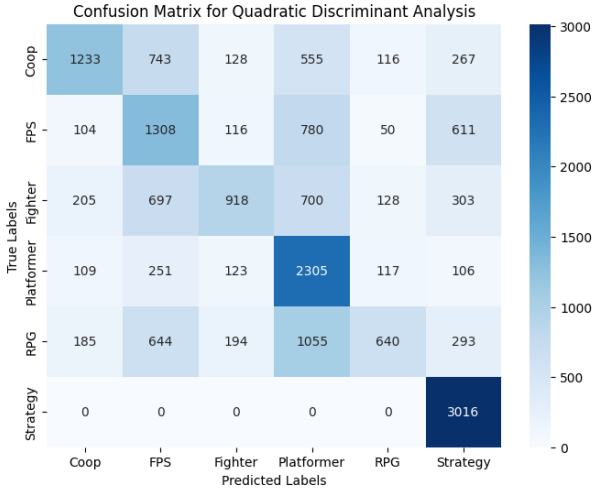
As above, certain performance parameters were chosen. This time only accuracy will be used to assess the performance, while also including runtime, loss and subset used for the relevant algorithms.

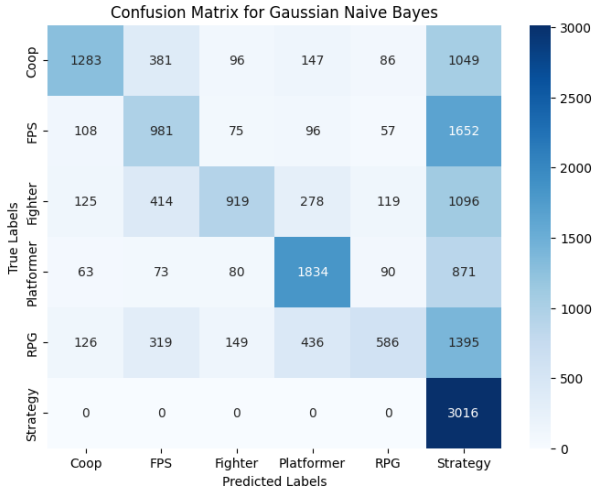
A series of Confusion matrices will follow, this time the loss and accuracy graphs for the tensor models will be missing. Why this is the case will be mentioned in the problems section. Lastly, a performance table will be created followed by a brief analysis.

8.1 Confusion Matrices



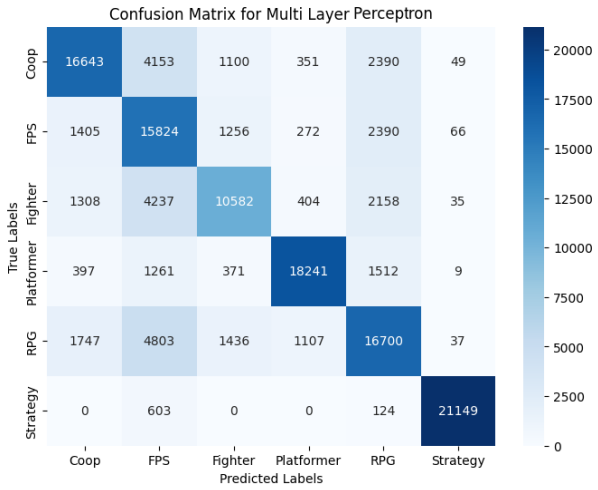


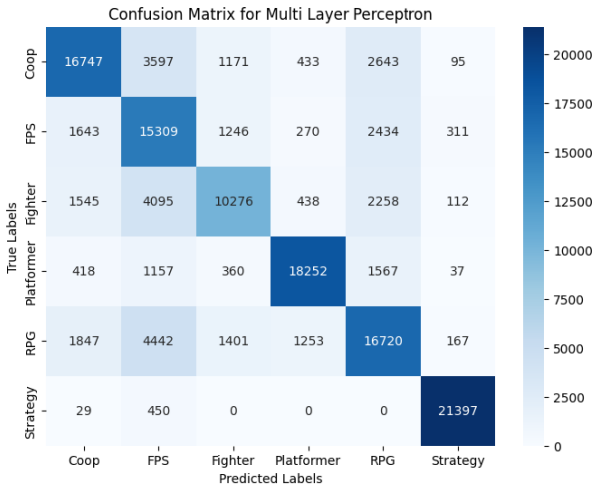


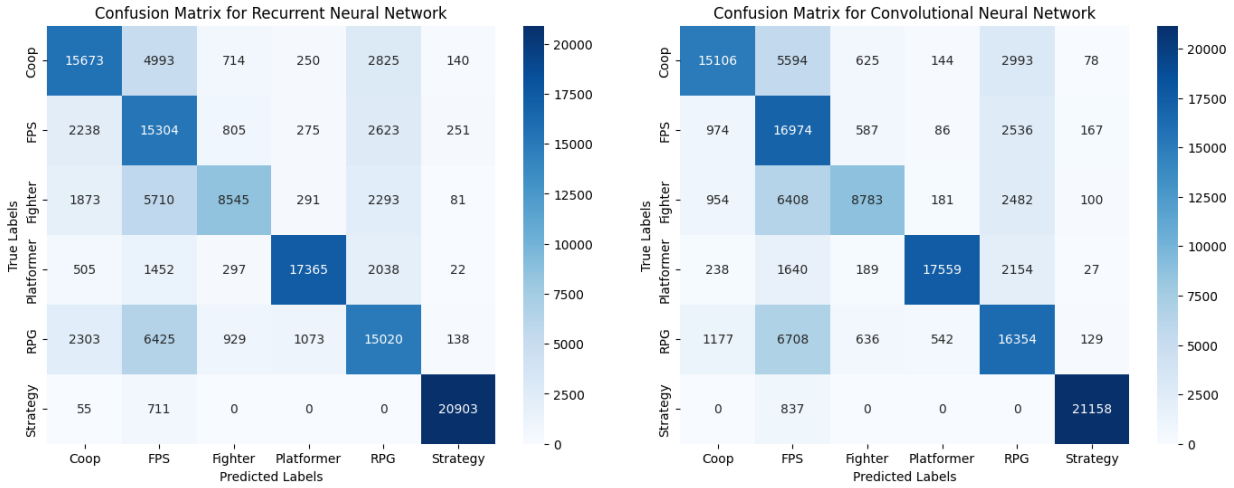


The following two figures had the following hidden layers assigned to them (from left to right):

1. (100)
2. (100,50)







8.2 Performance Table

Keep in mind that for the tensor models, the run time is missing. This is intended and will be discussed in the problem section.

Model	Accuracy	Runtime	Loss	Subset
Decision Tree	0.70	10min 7s	NaN	No
k-Nearest Neighbor	0.60	20min 10s	NaN	No
Logistic Regression	0.73	37min 30s	NaN	No
Random Forest	0.74	1h 43min 29s	NaN	No
Bernoulli Naive Bayes	0.55	6.78s	NaN	No
Quadratic Discriminant Analysis	0.52	32min 27s	NaN	Yes
Gaussian Naive Bayes	0.48	11.5s	NaN	Yes
Multi Layer Perceptron (100)	0.74	4h 37min 39s	NaN	No
Multi Layer Perceptron (100,50)	0.74	4h 49min 25s	NaN	No
Recurrent Neural Network	0.69	NaN	0.77	No
Convolutional Neural Network	0.72	NaN	0.70	No

8.3 Analysis

In comparison to the sentiment analysis, the performance has decreased significantly. The algorithms, where a subset had to be used, performed the worst, with the Gaussian Naive Bayes even falling below 50 percent.

Most models performed around the 70 percent accuracy mark, but there are a few outliers. The algorithms, where a subset was used, average around 48 percent. Bernoulli Naive Bayes also only managed 53 percent and the kNN achieved 61 percent. These models performed notably worse than the others.

The algorithms, where no subset was used, average around 69 percent accuracy. Something that is also worth mentioning is that the neural network models did not significantly outperform the non neural network models. For example the random forest as well as the logistic regression are able to keep up with the multi layer perceptron performance and even outperforming the tensorflows models.

Now of course, with more finetuning, the neural networks could potentially increase their accuracy, but the goal is not to create the best possible model, but just to compare the algorithms with generic parameters.

Runtimes were acceptable, but no comparisons will be made because of the missing values for the tensor models. Once again, the reason for it will be explained in the problem section.

9 Evaluation: Genre Classification without Keywords

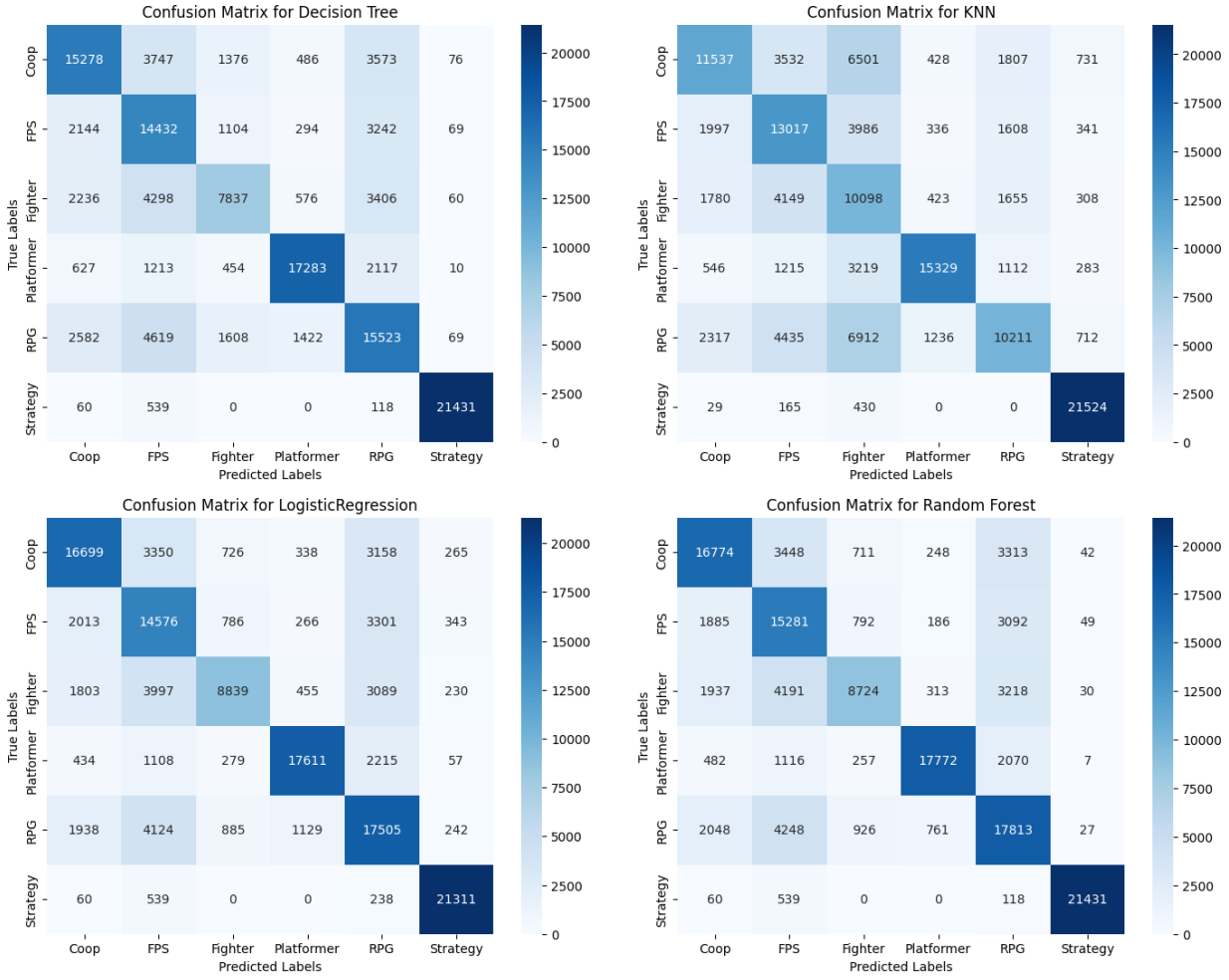
See the section before for the description. All the performance parameters stay the same, as well as the problems encountered during training.

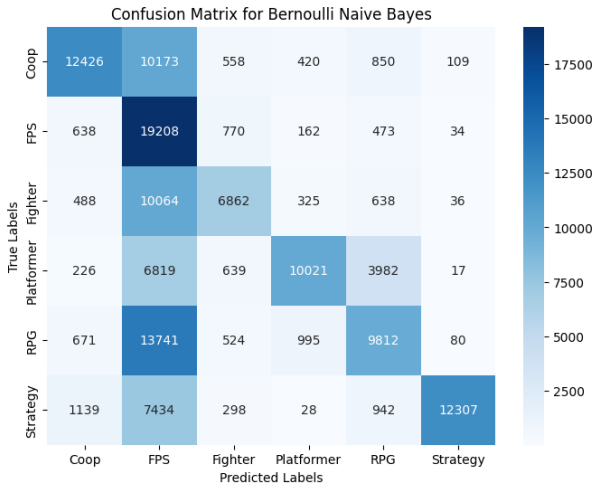
The keywords that were excluded are as follows:

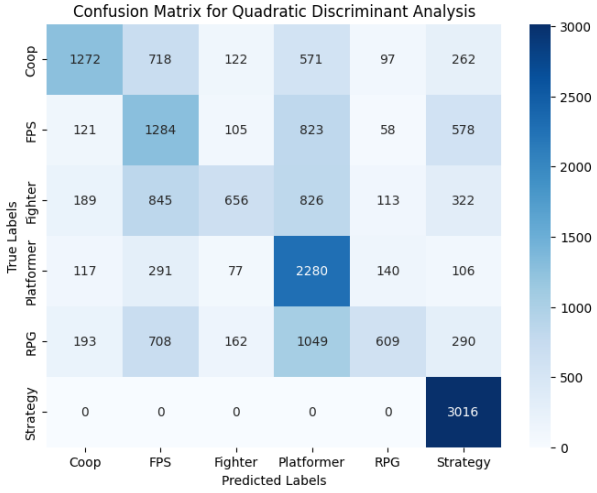
["rpg", "role playing game", "overwatch", "role", "shooter", "shoot", "fps", "platformer", "coop", "moba", "fight", "fighter", "baldur", "dota", "tekken", "fighterz", "ori", "persona", "helldiver", "skyrim", "ori", "smite", "payday", "pseudoregalia", "mortal", "kombat", "guilty", "gear", "brawlhalla", "strategy", "warhammer"]

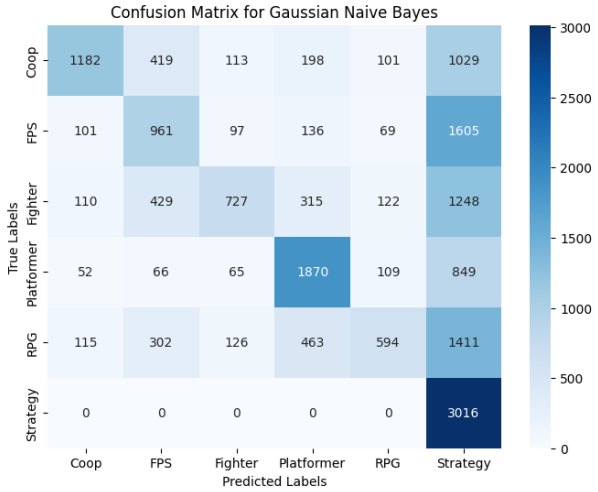
All these words either directly point towards the genre or are the names of the games that have been scraped for. These are all direct giveaways to the genre.

9.1 Confusion Matrices



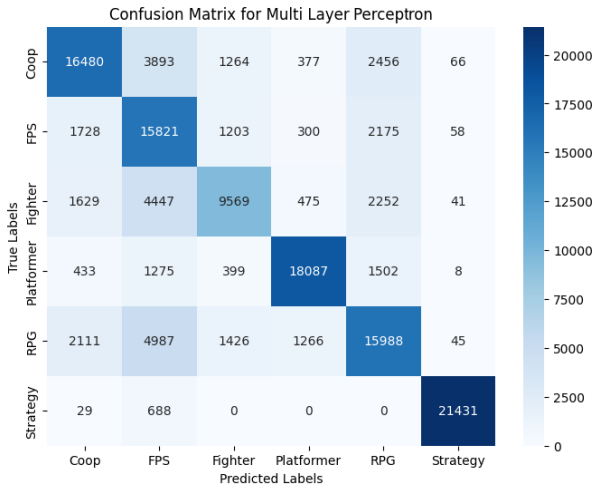


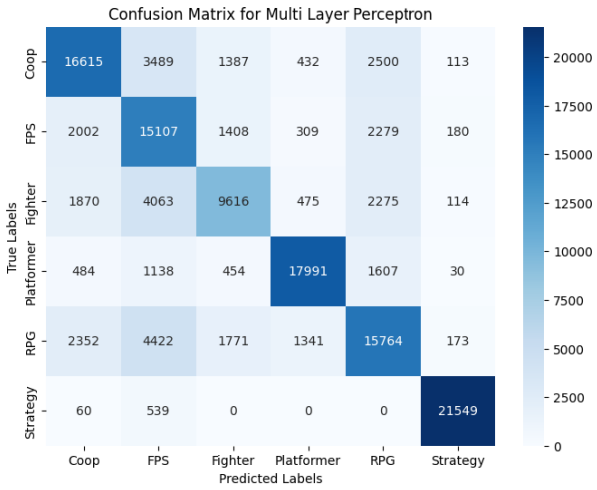


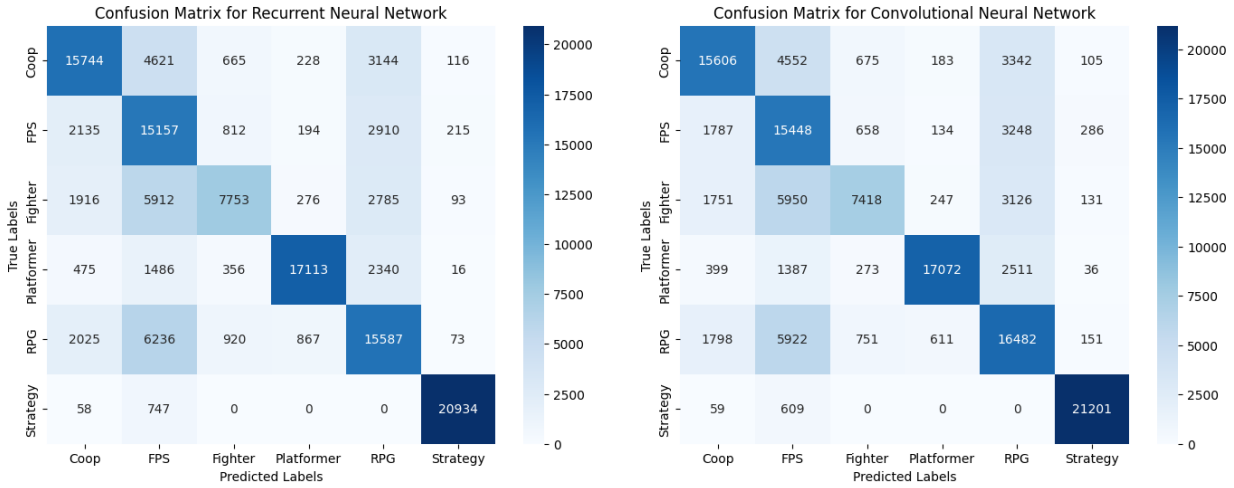


The following two figures had the following hidden layers assigned to them (from left to right):

1. (100)
2. (100,50)







9.2 Performance Table

Keep in mind that for the tensor models, the run time is missing. This is intended and will be discussed in the problem section.

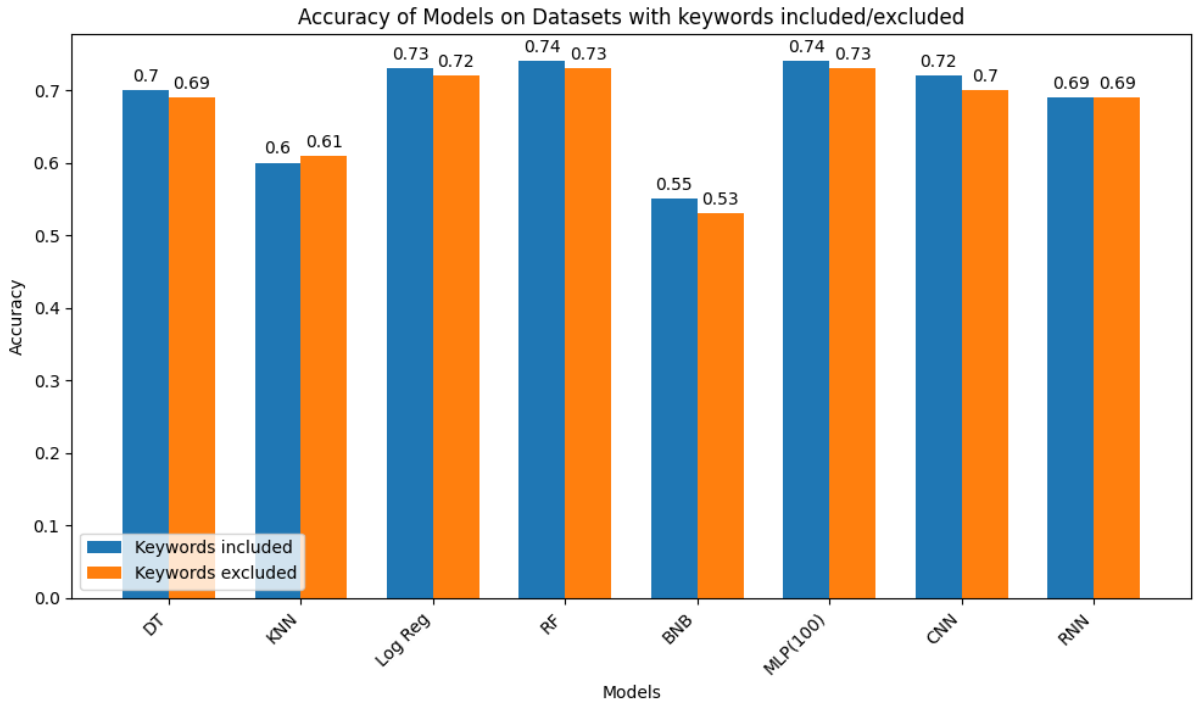
Model	Accuracy	Runtime	Loss	Subset
Decision Tree	0.69	10min 25s	NaN	No
k-Nearest Neighbor	0.61	19min 41s	NaN	No
Logistic Regression	0.72	40min 44s	NaN	No
Random Forest	0.73	1h 45min 58s	NaN	No
Bernoulli Naive Bayes	0.53	3.07s	NaN	No
Quadratic Discriminant Analysis	0.51	31min 37s	NaN	Yes
Gaussian Naive Bayes	0.46	11.4s	NaN	Yes
Multi Layer Perceptron (100)	0.73	4h 30min 4s	NaN	No
Multi Layer Perceptron (100,50)	0.72	4h 41min 5s	NaN	No
Recurrent Neural Network	0.69	NaN	0.77	No
Convolutional Neural Network	0.70	NaN	0.76	No

9.3 Analysis

Excluding the genre tags and the game names barely effected the predictive power of each model. All of the models are 2 percent points worse in terms of accuracy at most when compared with the model performance metrics in the previous section. The only odd one out is the kNN classifier, which was even able to improve its accuracy score by one percent.

This shows that the models do not rely upon the genre being mentioned or even the game title in order to accurately predict the genre of the game of the corresponding review.

Lastly, this graph shows the difference in accuracy for the graphs, where no subset was used. Furthermore, only the MLP with 100 hidden layers was included, because the difference in accuracy between the different layers is negligible.



10 Problems occurring during Coding and Training

This section will briefly touch upon the problems that were encountered when dealing with the different Machine Learning Algorithms.

10.1 Limited computational resources

When conducting this research, it was quickly apparent that certain algorithms as well as the amount of data can quickly crash the system[40]. In the case of this thesis, what plagued the progress the most was the limited allocated RAM. Sadly, no exact numbers are known, but the administration team allocated more RAM to the virtual server, which fixed a lot of the freezing and crashing that would occur and allowed for smooth progress. Symptoms of the limited resources included: crashing, instability and slow learning rate.

10.2 Instability

Tensorflow was especially prone to instability. With the ever growing list of CUDA and CUDnn versions, tensorflow versions as well as other dependencies, it is apparent that sometimes, the software is bound to be unstable. The symptoms of this instability were that the training would get stuck. The only way to remedy this was to restart the kernel and start the training from the beginning. After countless failed tries to fully train a model, it was decided to introduce a checkpoint system.

Tensor provides a checkpoint system, where after each epoch of a model, the model weights are being saved. If the training gets stuck or crashes for any reason, you can recall the last checkpoint and resume training from there. This allows to counteract the aforementioned problem of the training getting stuck.

The problem with this solution is, that providing the runtime for a training interval is not helpful. Depending on the times being stuck, the times for each successful training iteration will diverge significantly. Sometimes the training would get stuck during the first epoch, which means that no progress was made. This is why it was decided to exclude the runtime variable for the tensorflow models in the genre classification part.

Lastly, the reason the loss and accuracy graphs of the tensor models were missing in the genre classification being that the checkpoint system does not save loss and accuracy values for each epoch, meaning the values would have to be saved manually for each epoch and then had to be graphed in this way. It was decided to simply exclude these graphs than to go through all the effort and simply provide the accuracy and loss values for the last epoch.

11 Conclusion and further research

To summarize the findings in order to answer the research questions:

1. Sentiment scored 88 percent accuracy on average when looking at the performance of models which did not use a subset. Genre classification scored 69 percent. For more details, consult the corresponding sections.

2. For sentiment, most algorithms perform well, with most results being close together. There were a few outliers, but even those managed to predict reliably. For genre classification, the performance of the algorithms can generally be classified as good. There were 2 groups, 1 group of algorithms reaching accuracy scores around 70 percent. The other group of algorithms performed notably worse, with scores around 50 percent. Consult the performance tables for details. Taking these average values, there is a significant difference between sentiment and multi class analysis. These findings are similar to the results from other papers as well [5] [6].

3. Depending on which algorithms one is comparing, one can say that neural networks perform better, but looking at it from a broader perspective, neural networks do not significantly outperform classic machine learning algorithms. Taking the genre classification for example, where the genre tags are excluded, the random forest algorithm even managed to outperform the tensorflow neural networks while having an equal accuracy with the multi layer perceptron algorithm. Of course, neural networks can be fine tuned to potentially increase its performance, but the aim of this paper is not to generate the best results, but to only compare models in general.

4. Excluding the genre tags from the dataset barely had an impact on the performance. The accuracy stat at worst dropped by 2 percent. Consult the graph in the 9.3. analysis section for a visual representation of the findings.

As steam receives more and more data each day, the dataset size can be increased without problems by utilizing the code in the git repository. This data can be used for educational purposes, as long as it adheres to the steam subscriber agreement: https://store.steampowered.com/subscriber_agreement/#:~:text=You%20are%20entitled%20to%20use,others%20without%20the%20prior%20written.

References

[1] Scikit-learn Naive Bayes. https://scikit-learn.org/stable/modules/naive_bayes.html. Accessed: 12.04.2024.

[2] Naman Agarwal, Satyen Kale, and Julian Zimmert. Efficient methods for online multiclass logistic regression. In *International Conference on Algorithmic Learning Theory*, pages 3–33. PMLR, 2022.

[3] NJ Apao, LS Feliscuzo, CLCS Romana, and JAS Tagaro. Multiclass classification using random forest algorithm to prognosticate the level of activity of patients with stroke. *International Journal of Scientific & Technology Research*, 9(04):1233–1240, 2020.

[4] Christopher M. Bishop. Neural networks for pattern recognition. 1995. Available at: https://api.semanticscholar.org/CorpusID:60563397.

[5] Mondher Bouazizi and Tomoaki Ohtsuki. Sentiment analysis: From binary to multi-class classification: A pattern-based approach for multi-class sentiment analysis in twitter. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6, 2016.

[6] Mondher Bouazizi and Tomoaki Ohtsuki. Multi-class sentiment analysis on twitter: Classification performance and challenges. *Big Data Mining and Analytics*, 2(3):181–194, 2019.

[7] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984. Available at: https://books.google.at/books?id=JwQx-WOmSyQC.

[8] Kajal Chandani. An experimental comparison of deep lstm and grus for event classification in sports. 2019. Available at: https://arno.uvt.nl/show.cgi?fid=149699.

[9] B. Efron. Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, 7(1):1 – 26, 1979.

[10] Crammer et al. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[11] Kyunghyun Cho et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing*, 2014. Available at: https://api.semanticscholar.org/CorpusID:5590763.

[12] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[13] Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[14] Chris Fleizach and Satoru Fukushima. A naive bayes classifier on 1998 kdd cup. *Dept. Comput. Sci. Eng., University of California, Los Angeles, CA, USA, Tech. Rep*, 1998.

[15] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[16] Zhiyuan He, Danchen Lin, Thomas Lau, and Mike Wu. Gradient boosting machine: A survey, 2019. 1908.06951 Available at: https://arxiv.org/abs/1908.06951.

[17] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[18] Zainab Iqbal and Manoj Yadav. Multiclass classification with iris dataset using gaussian naive bayes. *International Journal of Computer Science and Mobile Computing*, 9(4):27–35, April 2020.

[19] Heba Ismail, S. Harous, and Boumediene Belkhouche. A comparative analysis of machine learning classifiers for twitter sentiment analysis. *Research in Computing Science*, 110:71–83, 12 2016.

[20] Vaibhav Jayaswal. Understanding naive bayes algorithm. Towards Data Science, 8th November 2020, last accessed: 6th June 2024, Available at: https://towardsdatascience.com/understanding-na%C3%AFve-bayes-algorithm-f9816f6f74c0.

[21] Hajer Kamel, Dhahir Abdulah, and Jamal M Al-Tuwaijari. Cancer classification using gaussian naive bayes algorithm. In *2019 international engineering conference (IEC)*, pages 165–170. IEEE, 2019.

[22] Yoon Kim. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing*, 2014. Available at: https://api.semanticscholar.org/CorpusID:9672033.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

[24] Vrushali Y Kulkarni and Pradeep K Sinha. Random forest classifiers: a survey and future research directions. *Int. J. Adv. Comput*, 36(1):1144–1153, 2013.

[25] Michael P. LaValley. Logistic regression. *Circulation*, 117(18):2395–2399, 2008.

[26] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[27] Wei-Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1:14 – 23, 01 2011.

[28] Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[29] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. Multilayer perceptron: Architecture optimization and training. 2016. Available at: https://www.ijimai.org/journal/bibcite/reference/2523.

[30] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[31] Bahador Saket, Alex Endert, and Cagatay Demiralp. Task-based effectiveness of basic visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 25(7):2505–2512, 2019.

[32] B.P. Salmon, W. Kleynhans, Colin Schwegmann, and J. Olivier. Proper comparison among methods using a confusion matrix. pages 3057–3060, 07 2015.

[33] Krizanic Sebastian. Bachelor Thesis Steam Reviews Classification. https://github.com/F34R7/Comparing-different-ML-Algorithms-on-Sentiment-Analysis-and-Genre-Classification-of-Steam-Reviews, 2024.

[34] Santosh Srivastava, Maya R Gupta, and Béla A Frigyik. Bayesian quadratic discriminant analysis. *Journal of Machine Learning Research*, 8(6), 2007.

[35] Statista. Video games - worldwide | statista market forecast. https://www.statista.com/outlook/dmo/digital-media/video-games/worldwide, Year the data was accessed. Accessed: April 10, 2024.

[36] Jiang Su and Harry Zhang. A fast decision tree learning algorithm. In *Aaai*, volume 6, pages 500–505, 2006.

[37] Jingwen Sun, Weixing Du, and Niancai Shi. A survey of knn algorithm. *Information Engineering and Applied Computing*, 1, 05 2018.

[38] Valve. Steam database. https://steamdb.info/. Retrieved from SteamDB Website.

[39] xPaw. Get reviews. https://partner.steamgames.com/doc/store/getreviews. Retrieved from Steam Partner Website.

[40] Zhi-Hua Zhou. Learnability with time-sharing computational resource concerns, 2024. Available at: https://arxiv.org/html/2305.02217v4.