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:	h0163938
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: 1st 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.

1 Introduction

Gaming is an ever-growing sector, the revenue rivaling even the movies and music industry [25]. 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 accessed by using the provided Steam API. Thanks to this policy, getting a large amount of labeled data poses no problem and thus allows this thesis to exist in the first place.

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

How do various machine learning algorithms perform when classifying the sentiment and genre of steam reviews?

How do the performances of these algorithms compare to each other? Is there a significant difference between performance on sentiment and multiclassing problems?

Is there a recommendation for an algorithm classifying game reviews?

Is there a significant difference between classic machine learning and neural networks?

How does multiclass 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.

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[23].

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 that are 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 the 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 being added manually for each game which is being 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, but please keep this fact in mind when considering the chosen genres.

Once the data has been correctly labeled, it is being 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 is 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 includes removing stopwords as well as lemmatization in order to boost performance. NLTK was used to for these steps. After successfully cleaning the reviews, they get tokenized.

Furthermore, 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.

[1] The steam database website: [28]
[2] API Description: [29]

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 sadly unbalanced in the sentiment department, because the focus lied within balancing out the genres, which evidently proved to be more successful.

There are also Reviews from the MOBA genre in the dataset, but sadly it was not possible to scrap enough reviews of that genre, since there are a very limited amount of games falling under that genre on steam, thus resulting in barely enough reviews. This is why the reviews of the MOBA genre have been neglected.

3 Sentiment Analysis: Packages and Algorithms

This section will talk about 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[8]. Furthermore, here is a brief overview of the algorithms and why they were chosen:

3.1.1 Naive Bayes

Naive Bayes for chosen for its simplicity and 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.

2 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[14].

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[14].

This allows for a runtime of O(d*c), where d is the number of features and c the number of classes[14].

This paper, written by Chris Fleizach, did not have good experiences with an increasing amount of samples, as it decreased accuracy, but increased recall interestingly. It will be interesting to see if the large dataset negatively impacts the accuracy of this model relative to the other tested models [9].

The second type of chosen algorithm is the Complement Naive Bayes. The reason this was chosen was:

> "CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the complement of each class to compute the model's weights."[1]

The dataset for this thesis does feature a significant imbalance in the sentiment department, thus making CNB a solid choice.[3]

3.1.2 Logistic Regression

Logistic Regression will be explained on the basis of the following paper, written by Michael P. LaValley [19].

Logistic Regression is a statistical method analyzing data and predicting a binary outcome. The paper claims that logistic regression is better suited for predicting binary

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

outcomes than linear regression because linear regression does not take into account the fact that the outcome variable is limited to two values. Odds ratios are a common way to express the results of logistic regression. Basically 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 naturally made for binary tasks. The runtime of logistic regression equals to O(d), where d is the number of features. This results in linear scaling.[4]

3.1.3 K-nearest neighbours

As described in this paper by Jingwen Sun et al, kNN is a simple yet effective algorithm [27].

kNN is non parametric and supervised, 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 basic steps are:

- 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 runtime is O(n log n) and thus begins to be computationally intensive with larger datasets.[5]

3.1.4 Random Forest

The basis for the description of this algorithm stems from this paper by VY Kulkarni et al[18].

Basically, the algorithm works as follows:

- Randomly select a subset of the training data with replacement to create a so called bootstrap sample.
- Construct a tree for each sample of the bootstrap. This means each tree stems from a different bootstrap sample.
- Randomly select a subset of features to consider for each tree

[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

- Combine the predictions from each tree in order to decide on the outcome. Majority is used for classifcation, while for regression averaging is used.
- With the combined predictions, a final decision is being made.

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

Lastly, the runtime efficiency of this algorithm is O(t * m log n), where as t is the number of trees, m is the number of features and n is the number of training examples.[6]

3.1.5 Passive Aggressive Classifier

Described by Crammer et al [5], this algorithm follows 2 rules: If the prediction is correct, the model tends to incorporate small to no updates, thus this 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.

The runtime for this algorithm is O(d), where d is the dimensionality of the input features. This is because the update step only requires a dot product and a scalar multiplication (because the algorithm deals with vectors).

The paper suggests a strong theoretical guarantee, so it will be interesting to see if it holds up in this thesis![7]

3.1.6 Multi-Layer Perceptron (Neural Network)

This paper by Ramchoun offers great insight of the innerworkings of the MLP algorithm [21].

The summary of the paper and how the algorithm works is as follows:

- 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.
- Performance is significantly impacted by the amount of nodes in each hidden layer and the amount of hidden layers employed. The paper proposes a mixed-integer non-linear optimization approach.
- The hidden layers basically assign weights which then influence the outcome. More does not always mean better in this case, since too many nodes may be prone to overfitting as well as being computationally unfeasible.

[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

- By using the approach suggested in the paper, the authors managed to achieve a higher classification accuracy in the legendary iris dataset compared to other neural network models.
- What is also interesting to mention, the paper managed to optimize the model while also lowering the amount of connections needed.

Defining a runtime in this case is difficult, because this type of algorithm needs to converge in order to determine the "best" model. In order to converge, the model needs several iterations. These iterations wary on many factors, like dataset size, hidden layer structure and just randomness. This is why defining a runtime efficiency variable is not helpful.

Nevertheless, neural networks allow models to learn about complex relationships and long term dependencies.[8]

3.1.7 Gradient Boosting Classifier

This survey looks at many types of different gradient boosting algorithms and serves as the foundation of understanding this type of algorithm[10].

The official sci-kit site defines the algorithm like this:

> "This algorithm builds an additive model in a forward stagewise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n classes regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only a single regression tree is induced."[8] [9]

GBA has 3 key components:

- Loss Function: This is also used in the neural networks in tensor. This is the primary objective of these algorithms. It measures how well the model predicts the target variable. One can quickly notice if the model is overfitting or not from this variable. The most common loss functions are usually MSE or Cross-Entropy.
- The weak learners: This is 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[10].
- The core of the GBA is the additive model. It combines the predictions of multiple weak learners to create a strong predictive model[10]. As it is additive, each weak learner is trained to correct the errors of its predecessor.

Now the problem when facing this algorithm in this paper was that there was a 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.

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

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

Furthermore, the runtime for this algorithm is as follows: O(n*m*t log m), where n is the number of training samples, m the number of leaves in each decision tree and t the number of trees.

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. [7]

3.2.1 Recurrent Neural Network

These types of RNN usually use a Long Short-term Memory layer. The paper by Hochreiter [11] 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.

Basically, 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. So which information is being stored and which is to be discarded immediately.
- Output gate: This regulates which information is to be put out, 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.

This allows for efficient computation, because the computational efficiency per time step is O(1). 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 that is tackling the the problem mentioned above are Gated Recurrent Units. Cho presents in his paper [6] alternatives to the LSTM layer. GRU are simpler and yet perform nearly as well as the critically acclaimed LSTM layer.

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[4].

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

Tensor offers both layers. [10]

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

3.2.2 Convolutional Neural Network

To explain the foundational structure of CNNs, a paper by O'shea et al is going to be used [20].

Basically, 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.

Furthermore, during training, backpropagation and gradient descent is used to minimize the difference between prediction and true labels.

Based on the above information, it is not surprising that CNNs usually excel at image classification[17]. CNN is doing a fantastic job at identifying local patterns, such as contrasting borders in an image, but it has also shown its merit when handling textual data as shown in this paper by Yoon Kim [16]. [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, stem from the scikit-learn library[8].

4.1.1 Decision Tree

The decision tree algorithm implement in sci kit is based on the CART algorithm. This specific version is explained in this paper by Jiang Su et al [26].

A brief explanation on how it works:

- CART Binary Tree: It builds a binary tree by recursively partitioning the data based on a single feature. That tree is grown by finding the feature and threshold that best separates these classes. So each tree is suppose to to create optimal separation so that the algorithm can make accurate predictions[26].
- Classification: CART uses the so called Gini impurity as the splitting criterion. So basically 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[26].
- 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.
- 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 [26].

New data points are then being fed into these trees in order to generate a prediction. The time complexity for training is O(n log n) where n is the number of data samples. Predictions on the other hand follow a O(log n) runtime efficiency.

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

4.1.2 K-nearest neighbours

See section 3.1.3.[13]

4.1.3 Logistic Regression

See section 3.1.2. Logistic Regression can handle multiclassing as well as mentioned on the sci kit site. Another example of it's effectiveness for these types of tasks can be found

[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

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] sprung to mind, where random forests were successfully utilized in a healthcare setting. What was eye catching is that the RF classifier outperformed other algorithms in basically all usual score metrics. It 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 [13].

Basically, how this works:

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

That is pretty much it. It looks at each feature, assigns it a boolean value and predicts upon a feature being present or well, not being present. It is incredibly efficient, boasting a runtime efficiency of O(n), where n being the number of features in the dataset.

Furthermore, this quote from the sci-kit site describing the algorithm solidified the choice for including this algorithm:

> "The decision rule for Bernoulli naive Bayes is based on
>
> $$P(x_i \mid y) = P(x_i = 1 \mid y)x_i + (1 - P(x_i = 1 \mid y))(1 - x_i)$$
>
> which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature i that is an indicator for class y, where the multinomial variant would simply ignore a non-occurring feature." [8]

Since there will 2 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

QDA is a bit more complicated. The description that follows is based in the official paper that forms the foundation for the sci kit implementation by Srivastava et al [24].

The key characteristics are:

- Bayesian estimate: In order to estimate the parameters of the Gaussian distribution for each class, Naive Bayes is used.

[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

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

- 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 deal breaker for large datasets.
- QDA solves a quadratic equation to determine the decision boundary, which once again, is computationally expensive.

Now the runtime is O(n), where as n is the number of features in the dataset. This is because for each feature, the algorithm only needs to calculate the covariance matrix and quadratic function once.

The reason why in this thesis a subset has been used for this algorithm, 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]

4.1.7 Gaussian Naive Bayes

This paper does a good job overviewing the algorithm [15].

Once again, this algorithm has these attributes:

- Assumption that features are independend 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.

It is runtime efficient, having O(n), where as n is the number of features. Thanks to its absence of complex calculations in contrast to the algorithm above, it is efficient.

There is also a neat little paper talking about the GNB and the legendary Iris Dataset [12], which led to the decision to include this algorithm in this thesis.

Furthermore, this 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

This section will touch base with the 2 types of neural networks that will be employed for the genre classification, both provided by the tensorflow library. [7]

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

[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

4.2.1 Recurrent Neural Networks

See section 3.2.1. [20]

4.2.2 Convolutional Neural Network

See section 3.2.2. [21]

[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 them into training and test data. There were 2 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 spare and a dense version of the data.

For Tensor it was decided to use the in built 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 100k reviews. For the genre classification, the subset included 15k reviews per genre, which there were 6 genres, resulting in dataset with 90k reviews.

Otherwise, when an algorithm allowed to input weights then this was done as well. 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.

Lastly, for the training of the models within the tensor environment, the 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.

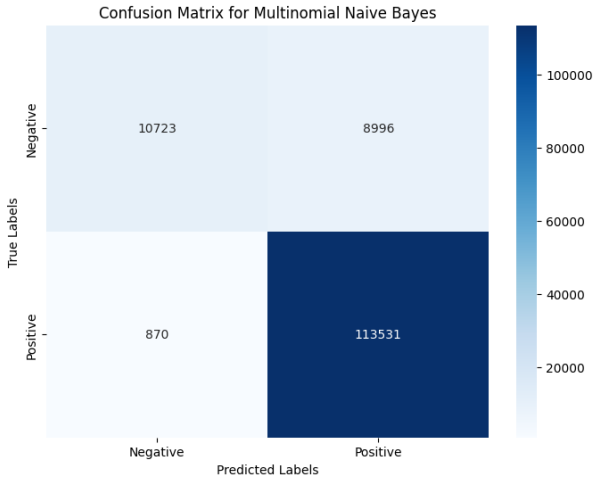
6 Evaluation: Sentiment Analysis

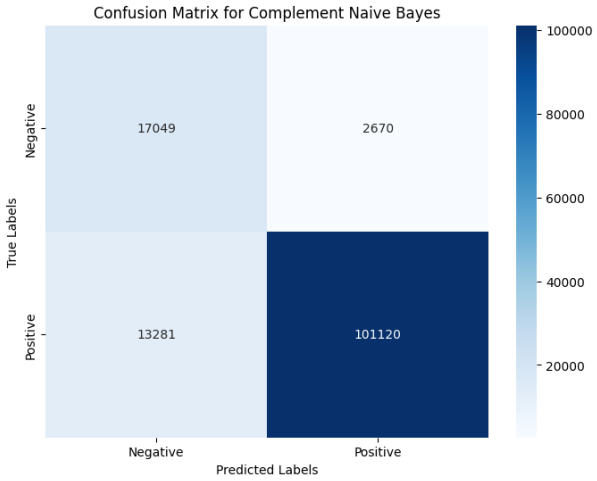
In order to evaluate the algorithms, certain performance parameters were chosen. Accuracy, Recall and precision will be used in the sentimental part to portray the performance of each algorithm. Additionally, runtime, loss and subset used are going to be included where applicable. Furthermore, a confusion matrix [22] 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 of the performance of these models.

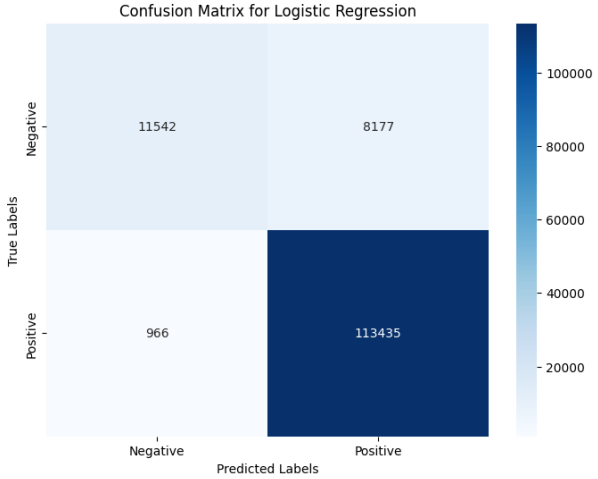
Runtime is measured with the inbuilt time command in jupyter. Accuracy is calculated with the inbuilt methods of both sci-kit and tensor. Loss is calculated in the tensor training session. For more details, please refer to the github repository and the sources within this paper and the notebooks.

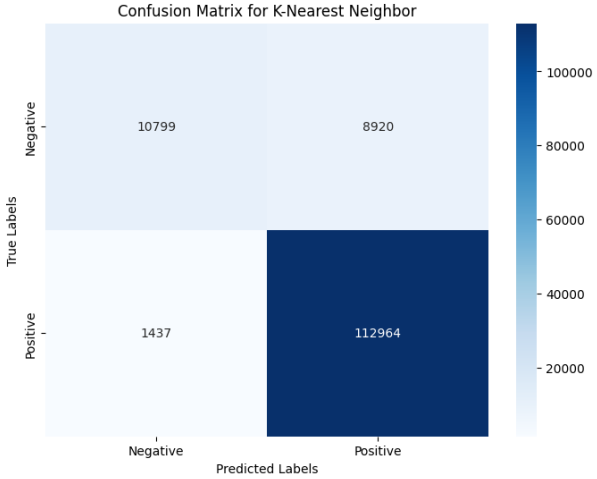
What follows is a series of Confusion Matrices, each labeled according to the algorithm it represents. Afterwards, a table will be presented with different performance variables. Lastly, an analysis will close the sentiment analysis chapter.

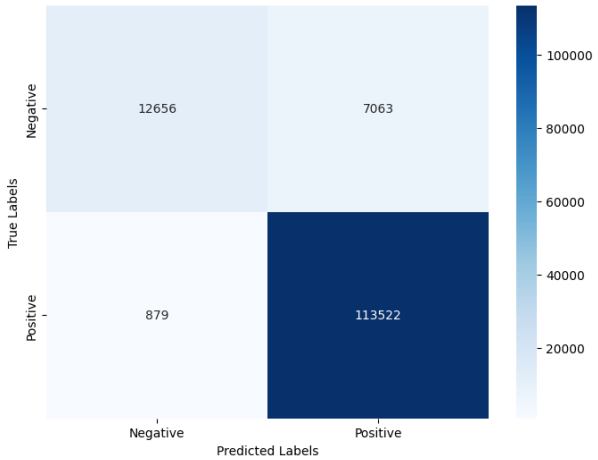
6.1 Confusion Matrices

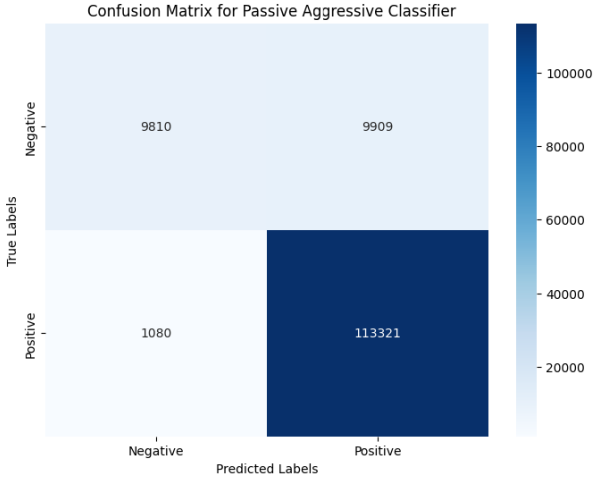

Confusion Matrix for Multinomial Naive Bayes
10723
8996
870
113531
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000

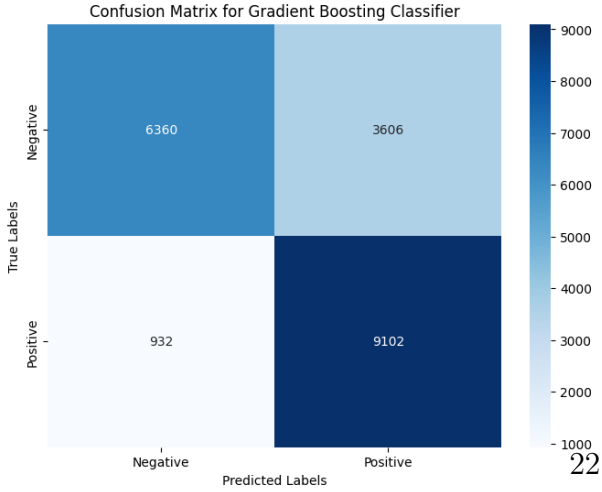

Confusion Matrix for Complement Naive Bayes
17049
2670
13281
101120
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000


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

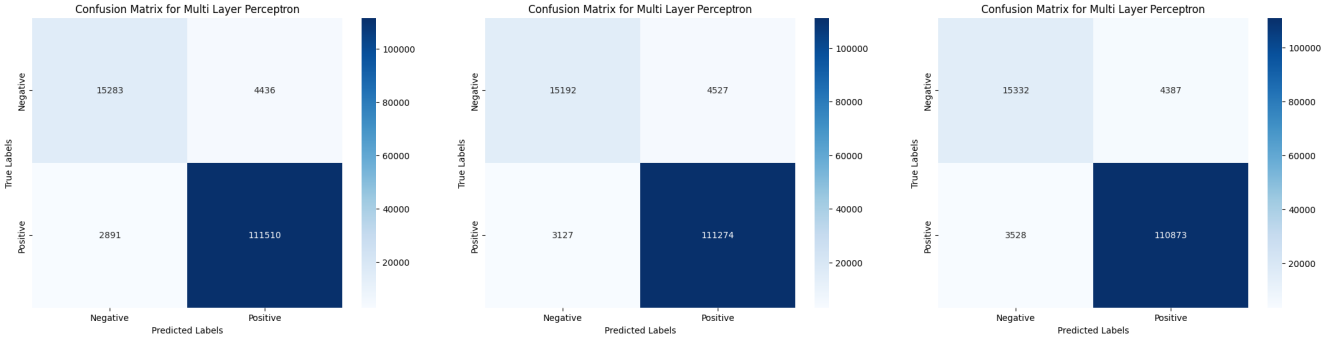

Confusion Matrix for Random Forest
12656
7063
879
113522
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000

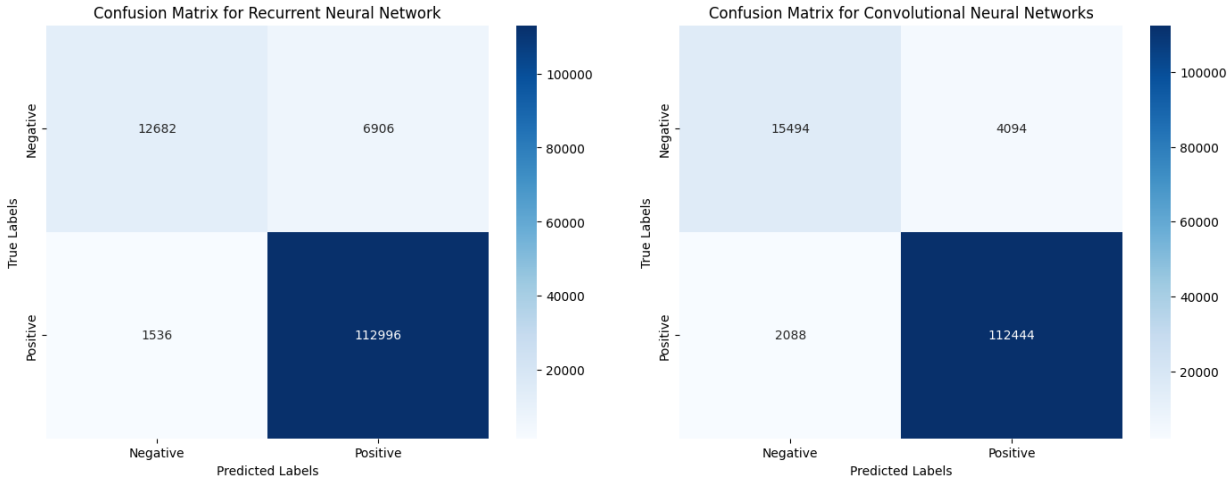

Confusion Matrix for Passive Aggressive Classifier
9810
9909
1080
113321
Negative
Positive
True Labels
Predicted Labels
100000
80000
60000
40000
20000


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

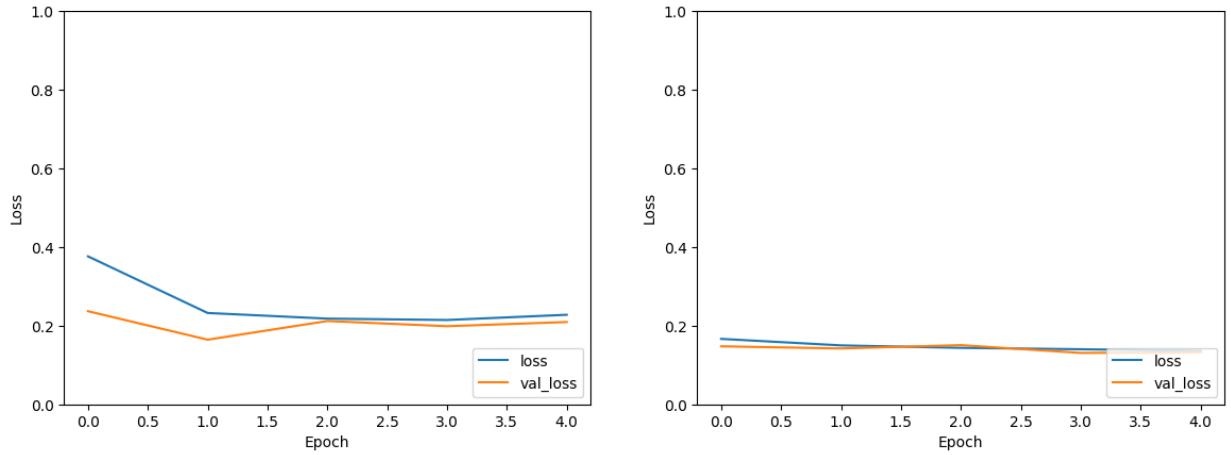
The following three figures had the following hidden layers assigned to them (from left to right):

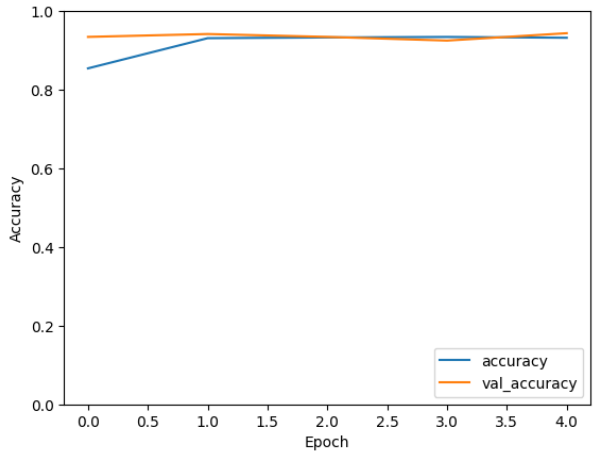
1. (100)
2. (100,50)
3. (200,100)

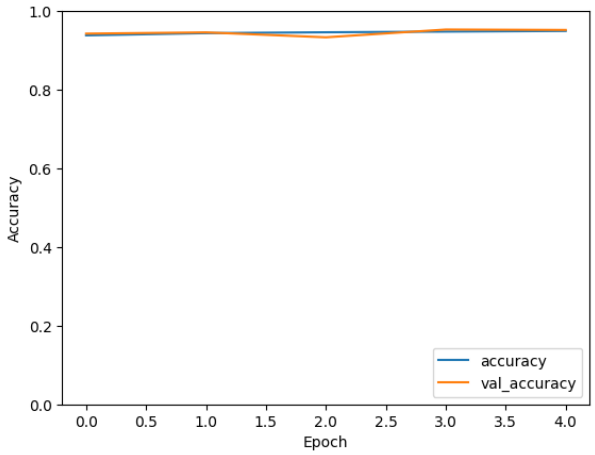




For these graphs, the left one always corresponds to the RNN and right one to the CNN.







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 has 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 though 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.

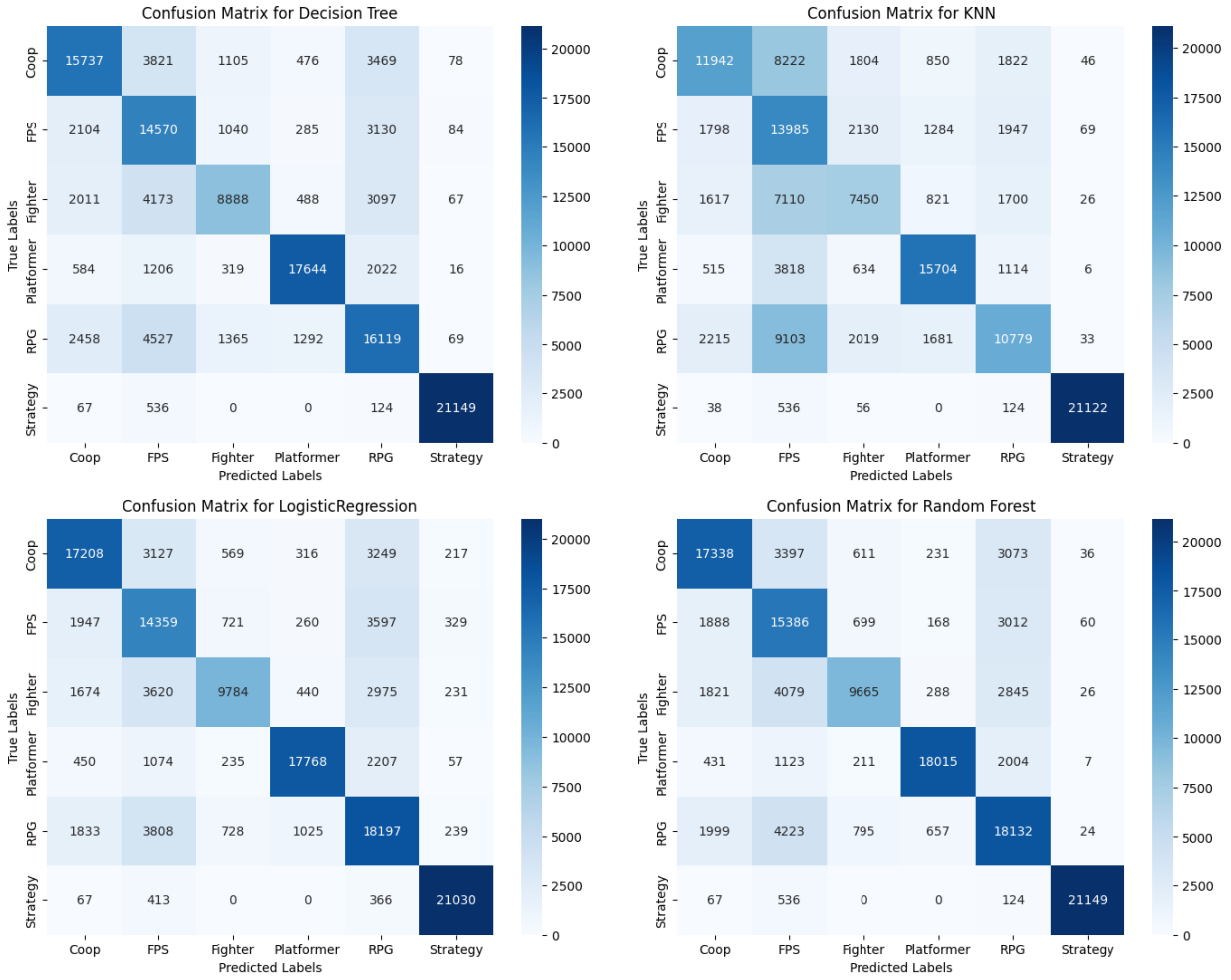
In summary, every model tested has proven to be effective at the given task.

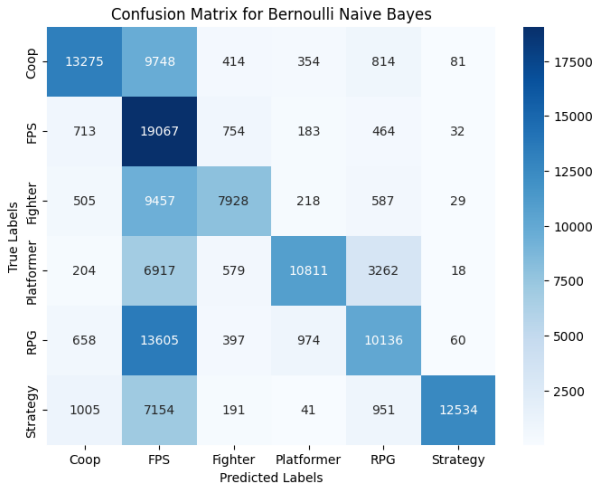
7 Evaluation: Genre Classification with Keywords

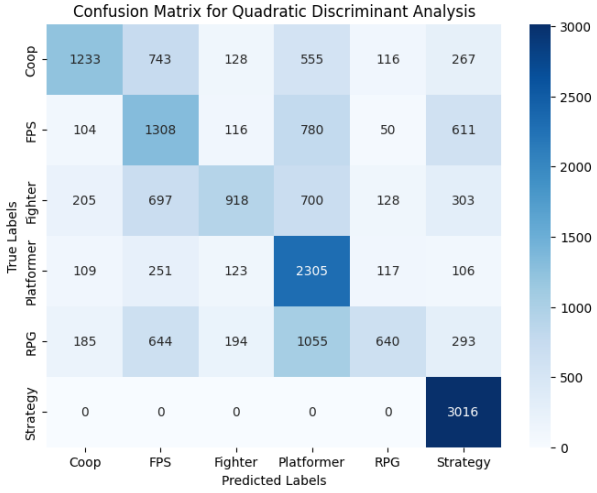
As above, certain performance parameters were chosen. This time only accuracy will be used to assess the performance, while also including runtime, loss and subset used for the relevant algorithms.

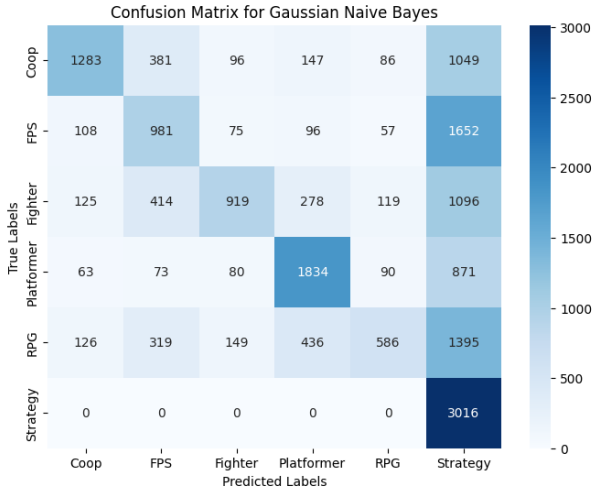
A series of Confusion matrices will follow, this time the loss and accuracy graphs for the tensor models will be missing. Why this is the case will be mentioned in the problems section. Lastly, a performance table will be created followed by a brief analysis.

7.1 Confusion Matrices



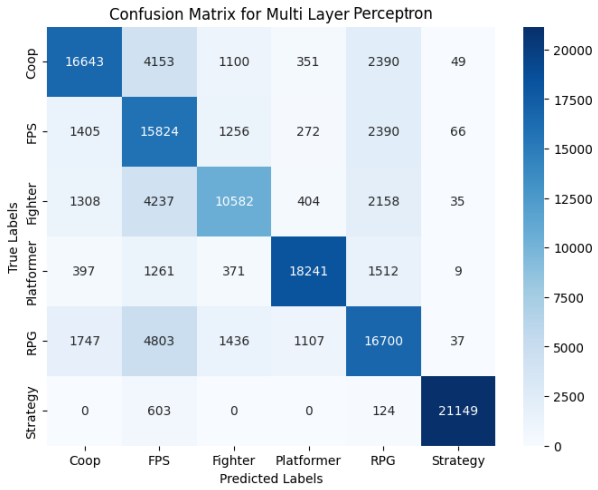


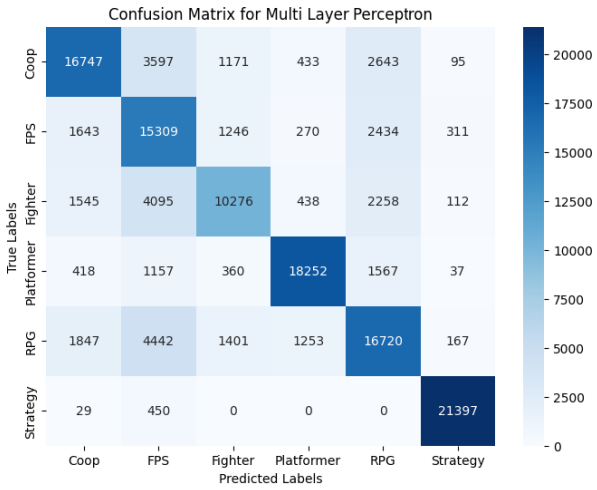


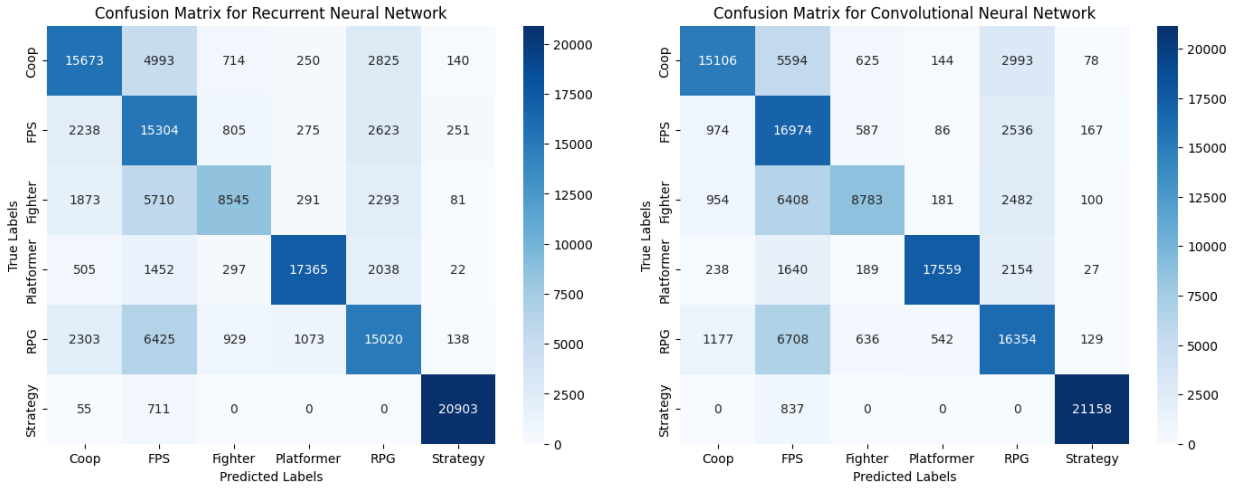


The following two figures had the following hidden layers assigned to them (from left to right):

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







7.2 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

7.3 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. What is also very interesting 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.

8 Evaluation: Genre Classification without Keywords

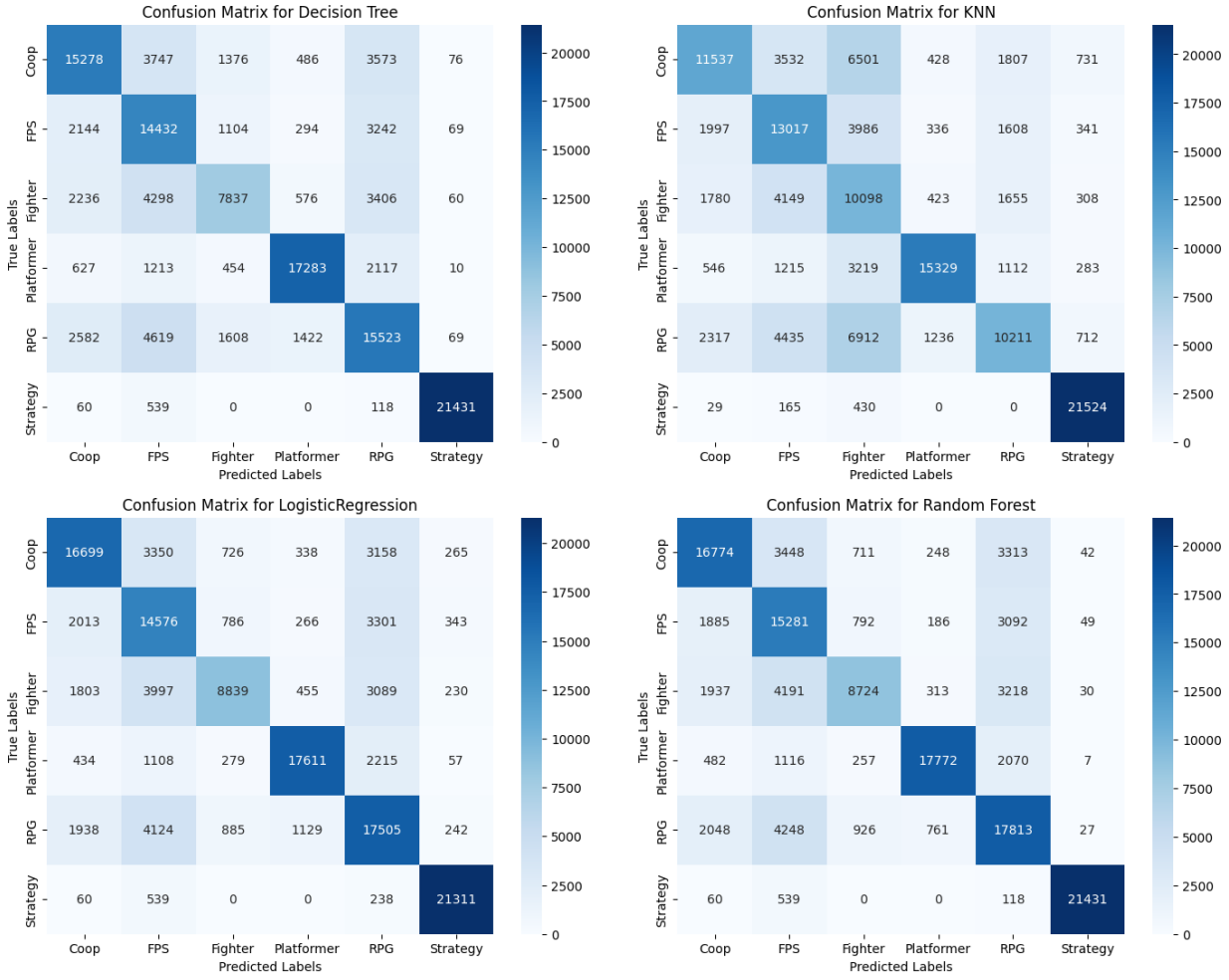
See the section before for the description. All the performance parameters stay the same, as well as the problems encountered during training.

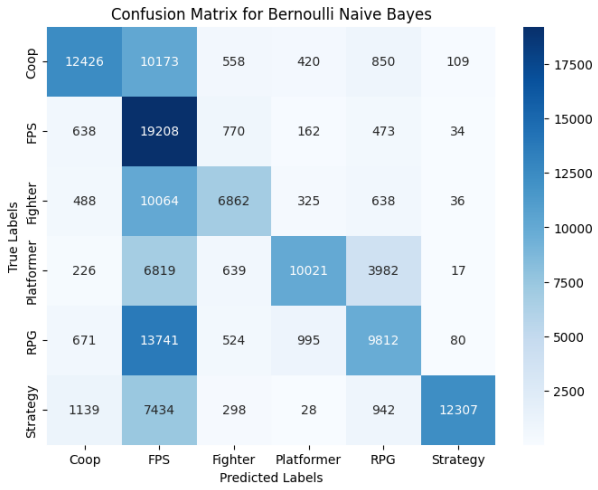
The keywords that were excluded are as follows:

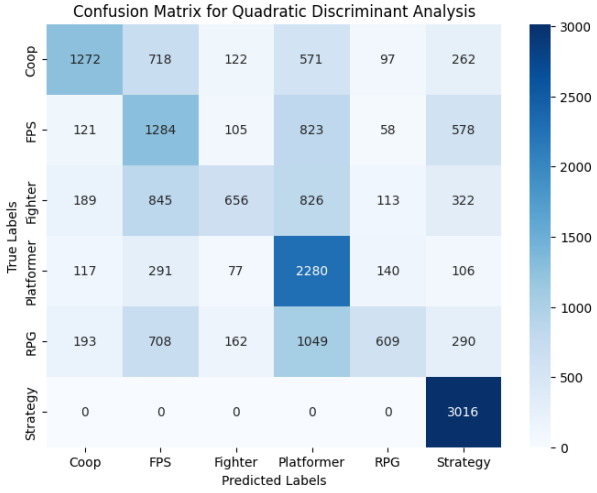
["rpg", "role playing game", "overwatch", "role", "shooter", "shoot", "fps", "platformer", "coop", "moba", "fight", "fighter", "baldur", "dota", "tekken", "fighterz", "ori", "persona", "helldiver", "skyrim", "ori", "smite", "payday", "pseudoregalia", "mortal", "kombat", "guilty", "gear", "brawlhalla",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

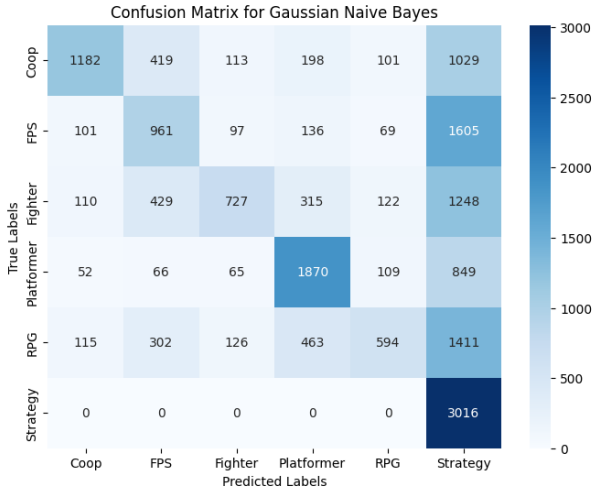
All these words either directly point towards the genre or are the names of the games that have been scraped for. These are all direct giveaways to the genre.

8.1 Confusion Matrices



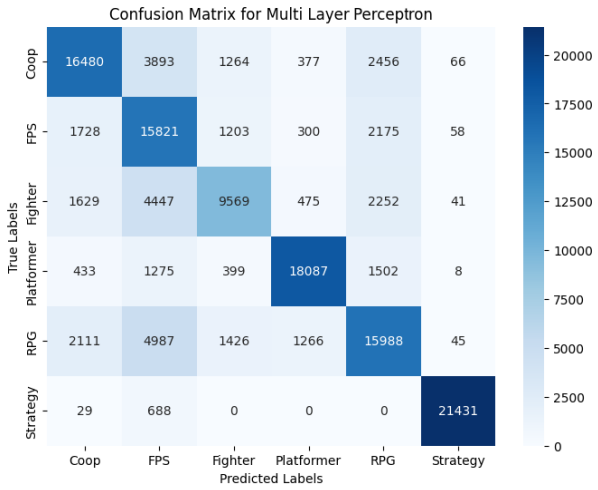


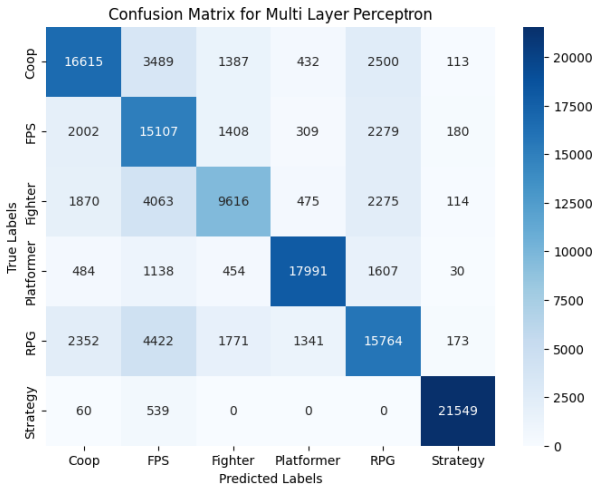


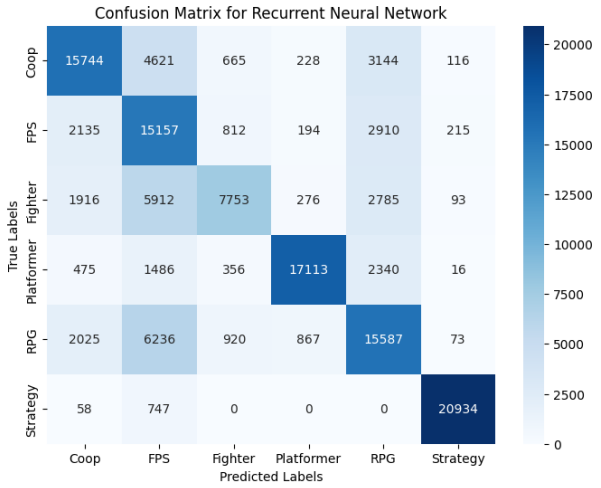


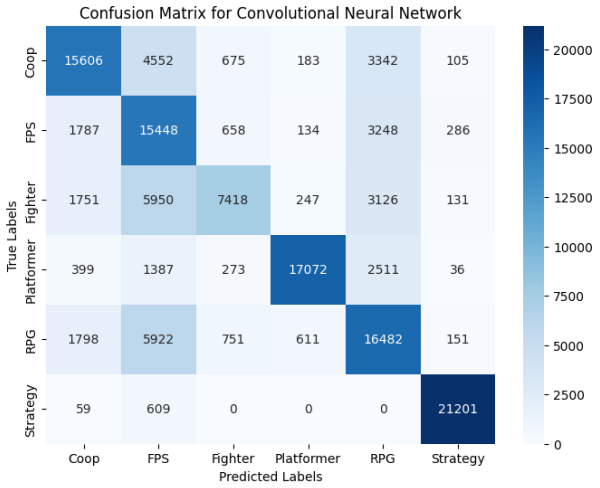
The following two figures had the following hidden layers assigned to them (from left to right):

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	0.69	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

8.3 Analysis

Excluding the genre tags and the game names had barely an effect on 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 1 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.

9 Summary of the Analysis

When it comes to the sentiment, the accuracy scores average above 90 percent for all models, with recall being especially high averaging around 98 percent. Genre classification performed significantly worse than its sentiment counterpart, but still managed to

achieve an impressive 69 percent accuracy, when only the models not using a subset are considered.

In the sentiment department, there were no notable differences in accuracy between the models. All of them performed well for the task given. For the genre classification, there were a few outliers where the accuracy stat was notably lower than their better performing counterparts. As mentioned above, genre classification performed worse accuracy wise than the sentiment analysis.

When it comes to excluding genre relevant tags as well as the game titles, the algorithms did not perform significantly worse, where the majority of models did worse by 2 percent accuracy at most.

There was no significant difference between classic machine learning methods and neural networks when it comes to accuracy. Runtime on the other hand was diverging. MLPs runtime was so high because it used the CPU and not the GPU to do its training. This is where tensor shines, if it would have been more stable at least. Otherwise the runtimes are as expected when compared to the runtime efficiencies.

When it comes to recommending an algorithm, one has to consider many things that are important for the person deciding which one to use. In this case, a lot of performance stats were provided in order to decide if there is a best fit for specific use cases. This thesis will refrain from offering a recommendation and rather point towards the data and let the reader decide for him/herself, what to choose.

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 bring a system to its knees[30]. 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 does not want to cooperate properly. The symptoms of this instability were that the training would simply 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 constant getting stuck.

Now 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. Not to mention that sometimes the training would get stuck during the very first interval! 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 very last epoch.

11 Conclusion and further research

Despite the problems, the research was concluded with satisfactory results. All questions have been answered and a good variety of algorithms have been tested to generate a decent overview.

As steam receives more and more data each day, the dataset size can be increased without problems by simply utilizing the code in the git repository. This data can be used in a variety of research, as long as it follows the license rules naturally.

This research can be expanded with more algorithms and maybe even serve as a foundation for new store features that incorporate these models.

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] Kajal Chandani. An experimental comparison of deep lstm and grus for event classification in sports. 2019.

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

[6] 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.

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

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

[9] 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.

[10] Zhiyuan He, Danchen Lin, Thomas Lau, and Mike Wu. Gradient boosting machine: A survey, 2019.

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

[12] 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.

[13] 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.

[14] Vaibhav Jayaswal. Understanding naive bayes algorithm.

[15] 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.

[16] Yoon Kim. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing*, 2014.

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

[18] 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.

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

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

[21] Hassan Ramchoun, Youssef Ghanou, Mohamed Ettaouil, and Mohammed Amine Janati Idrissi. Multilayer perceptron: Architecture optimization and training. 2016.

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

[23] 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.

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

[25] 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.

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

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

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

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

[30] Zhi-Hua Zhou. Learnability with time-sharing computational resource concerns, 2024.