Machine Learning Project

Classification of inappropriate language in game chats

STORAI Romain, FAURE Benoit, MATHIEU Antoine

April 5, 2023

Contents

1 Introduction

In today's world, online video games have become a popular way to socialize and have fun. However, the issue of inappropriate language in video game chat is a growing concern. The current method of detecting inappropriate language relies on using lists of banned words, which can be inefficient as it fails to take into account the context of the sentence. This means that inappropriate sentences without banned words may not be detected by the system. To address this problem, we propose a study of multiple machine learning models to classify inappropriate sentences. Our final goal is to propose a machine learning model that can accurately determine text appropriateness in online video game chat, and to integrate this model into the chat system in real-time to facilitate effective moderation. This will help to create a safer and more enjoyable environment for all players.

1.1 What is Inappropriate Language?

We define, for the following work, an *Inappropriate Language* as a word or a sentence that contains any of the following:

- **Profanity** - This includes any language that is considered vulgar, offensive, or obscene. This can include swear words, sexual language, and derogatory terms.
- **Hate speech** - Hate speech is language that is intended to demean, discriminate against, or incite violence or hatred towards a particular group of people based on their race, ethnicity, gender, religion, sexual orientation, or other characteristic.
- **Insults** - This includes any language that is intended to insult or belittle someone else. This can include name-calling, personal attacks, or derogatory comments about someone's appearance, abilities, or personality.
- **Threats** Threats are language that is intended to intimidate or harm another person. This can include physical threats, verbal abuse, or intimidation.

1.2 Problem

Our problem could be summarized as a supervised binary classification task, where the input data consists of text messages and the output labels indicate whether the message is appropriate or inappropriate.
We will evaluate some machine learning and deep learning models, and benchmark them using multiple metrics.

2 Study

Our study is a benchmark of several models, trained and tested on the same data. This benchmark uses each model on the same task and compare them with defined metrics. In this section, we describe the dataset used, including the preprocessing steps, as well as the models used. Additionally, we discuss the metrics we chose to evaluate model performance
While cross-validation, such as k-fold, is a common method for evaluating models, it may not be feasible for such a large dataset or complex models due to their long training times. This is way we went for the classic train-test-split, or hold-out, method. Our dataset was split into training, testing and validation data. And all the models were trained and tested on the same data. Please note that all the subsets are stratified because the proportion of Appropriate / Inappropriate classes is unbalanced.

2.1 Dataset

2.1.1 Details

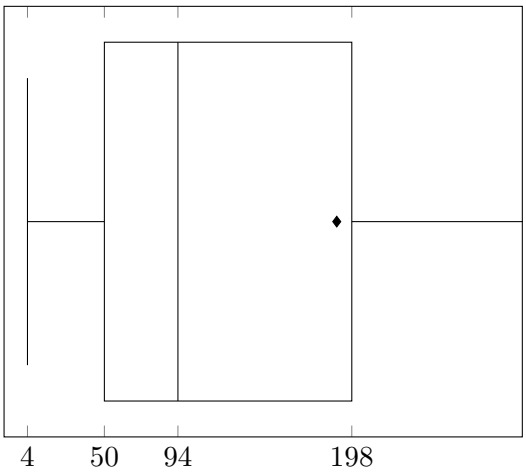
Our dataset contains texts associated with a class. The class can be Appropriate (0) or Inappropriate (1). The texts are in English and come from social media comments (Twitter, Reddit, Youtube and Wikipedia Comments). We are aware that this dataset will not be able to fully answer our problem given its source: video game chat texts are very different from social medias comments. However, we have not been able to find such data in the gaming domain.
We will describe bellow the creation of the dataset, but here is the global decriptive analysis:

We ended up with a dataset containing 519369 labeled texts (598929 before cleaning). The distribution of the classes is as follows:

Dataset	Appropriate	Inappropriate	Size
HSAOL	5.77%	94.23%	24783
Measuring HS	63.82%	36.18%	135556
Insulting	73.58%	26.42%	6594
Jigsaw	92.82%	7.18%	312735
Jibes & Delight	57.15%	42.85%	119261
Merged & Cleaned (our dataset)	**77.50%**	**22.50%**	**519369**

Table 1: Classes distribution of the final dataset

The merged and cleaned (methods described later) dataset contains a total of 519,369 labeled texts, with 77.50% of them being categorized as Appropriate and 22.50% as Inappropriate 1. This indicates that the dataset is imbalanced, with the majority of the texts being categorized as Appropriate.
It is important to note that the distribution of classes varies significantly across the source datasets, with some being heavily biased towards Inappropriate texts (e.g., HSAOL and Measuring Hate Speech) and others being more balanced (e.g., Jigsaw and Jibes & Delight). We are aware that it could potentially impact the performance of models trained on, but we couldn't find any better laternatives.
PS: The source datasets are described later.



(a) Boxplot

Description	Chars
min	4
max	4999
average	190
std	198
25%	50
median	94
75%	198

(b) Values

Figure 1: Length of texts

The figure 1 is a quick analysis of the texts contained in our dataset. Most of them are of small-to-medium length, which is close to the in-game chats length.
We thus obtain 103 MB of data, which we have separated into Train, Validation and Test data, representing respectively 60%, 20% and 20% of the total data. This split has been stratified, which allows each of these sets to have a class distribution similar to the starting set.

2.1.2 Creating the dataset

Our dataset is a merge of multiple datasets. For each of the models, we have selected characteristics that allow us to categorize the texts as appropriate or not. Moreover, we tried to vary types of inappropriate language: there are datasets of Hate Speech that target groups of individuals, as well as datasets of insults that target individuals directly. This will ensure that we do not bias our model with certain types of inappropriate language.
After gathering and merging the data, we cleaned the data to reduce the complexity of it:

- We lowercased all the characters,
- Removed unecessary English stop words,
- Tried to remove as much as non-english sentences as possible,
- Removed sentences over 5000 characters (11 in total),
- And lemmatized all words, which is the process of reducing a word to its base or dictionary form (lemma form).

2.1.3 Sources

Here are the 5 datasets we used to create our dataset. There are some details about each, such as the source, the content and the description of the dataset, any comments we had, and an explanation on how we matched the source dataset's labels to our Appropriate / Inappropriate label. It is difficult to find datasets that contain slurs and hate speech because they are highly controversial due to their potentially offensive content. However, their usage is strictly positive in text analysis.

1. **HSAOL - Hate Speech and Offensive Language**

   Source Github repository

   Datatype Tweets

   Description The text is classified as: hate-speech, offensive language, and neither.

   Comments This dataset contains mainly hate speech data and isn't really defined on *neither*.

   Merge process

Inappropriate	Offensive (1) or Hate Speech (0)
Appropriate	Neither (2)

2. **Measuring Hate Speech**

   Source Huggingface

   Datatype Social media comments (Twitter, Reddit, Youtube)

   Description 10 ordinal labels (sentiment, (dis)respect, insult, humiliation, inferior status, violence, dehumanization, genocide, attack/defense, hate speech), which are debiased and aggregated into a continuous hate speech severity score.

   Comments This dataset contains mainly hate speech data (targeting class of individuals, not individuals directly).

   Merge process

Inappropriate	hatespeechscore > 0.5
Appropriate	hatespeechscore ≤ 0.5

3. **Insults Dataset**

   Source Kaggle Competition

   Datatype Comments

   Description The label is either 0 meaning a neutral comment, or 1 meaning an insulting comment.

   Comments This dataset does not contain particular hate speech, but aims individual.

   Merge process

Inappropriate	Insult $= 1$
Appropriate	Insult $= 0$

4. **Jigsaw Dataset**

   Source Kaggle Competition

   Datatype Wikipedia Comments and Chats

   Description Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity are: toxic, severetoxic, obscene, threat, insult, identityhate.

   Merge process

Inappropriate	one of the class mentionned above
Appropriate	none

5. **Jibes and Delight**

   Source Paper

   Datatype Reddit Comments

   Description The texts are preprocessed and fetched from different reddit channels to classify them in two categories: insult and not.

   Comments Individual targeting.

   Merge process

Inappropriate	Insulting
Appropriate	none

2.2 Models

2.2.1 Preamble

To input our model, we require either words or sentences, which necessitates performing some natural language processing (NLP) to transform our text into a machine-readable format. To accomplish this, we employed a tokenizer and an embedder. The tokenizer divides the text into smaller units, such as words or characters, to make it easier to handle. After that, the text is converted into numerical vectors using an embedder. The embedder maps the text into a high-dimensional vector space, where each dimension corresponds to a unique feature of the text.
Initially, we employed a traditional tokenizer and embedder. Later, we experimented with a pre-trained embedder known as GloVE [6] trained on tweets to see if we could improve efficiency.
The following sub-sections showcase the different models we benchmarked on our dataset.

2.2.2 Decision Tree

Decision Tree is a supervised predictive model that can be used for both classification and regression tasks. In this note, we will discuss the modeling, learning process, and evaluation of Decision Trees.

1. **Modelling**
   A Decision Tree is a tree-like structure that represents a set of decisions and their possible consequences [12]. Each node in the tree represents a decision, and each branch represents a possible outcome of that decision. The leaves of the tree represent the final outcomes.

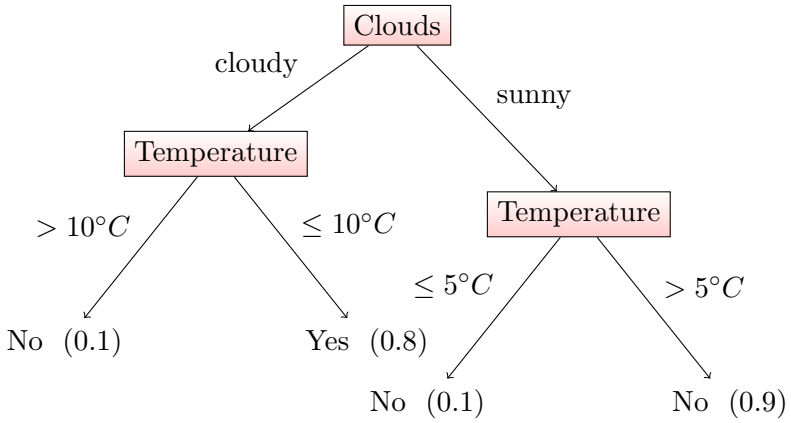


Figure 2: Example decision tree for predicting rain based on the clouds and temperature.

   The above figure details an example of decision tree that predicts rain according to two features: Clouds and Temperature. The root node is the clouds variable, with possible values of sunny or cloudy. Depending on the clouds, we then split the data based on the temperature conditions, until we arrive at a leaf node that provides the predicted outcome and eventually associated probability.
   Decision trees are often used in data science due to their simplicity and the possibility of drawing them.
   But how are the features chosen on each node? How the splitting process works? And more globally, how do we grow such a tree?

2. **Learning Process**
   Some Decision Tree Algorithms exist. Globally, they all follow the process we will describe,

but we will deepen the comprehension using the CART [7] algorithm (Classification and Regression Tree algorithm) since itś the one used in scikit-learn [5]
Growing a tree consists of a recursive algorithm that performs two main steps:

- **Feature Selection:**
  The algorithm selects the best feature to split the data at the current node, based on a chosen impurity measure such as entropy or Gini index. The feature with the highest information gain or the highest reduction in impurity is chosen as the splitting feature. This process is repeated recursively for each child node until a stopping criterion is reached (e.g., a minimum number of samples per leaf node, or a maximum depth of the tree).
- **Finding Threshold:**
  After selecting the splitting feature, the algorithm determines the optimal threshold value for that feature to split the data into two or more groups based on the feature's values. The threshold value can be determined by evaluating various possible threshold values for the feature and selecting the one that results in the best split based on the chosen impurity measure. This process of finding the optimal threshold is repeated for each feature and each node until a stopping criterion is reached.

But when should the recursion stop?
For few features data, it is easy to imagine that after few recursions the remaining subset of features would end empty. However, it is common to encounter more complex data with much more features. Going along the whole features could overfit the model to the training data. That is why the decision tree learning is controlled by a stopping criteria. There are several common stopping criteria that can be used:

- Minimum number of samples: Stop growing the tree if the number of samples in a node is less than a certain threshold. This helps prevent overfitting to noise in the data.
- Maximum depth: Stop growing the tree if the maximum depth of the tree has been reached. A shallow tree can be more interpretable and generalizable than a deep tree, which may be more prone to overfitting.
- Minimum reduction in impurity: Stop growing the tree if the reduction in impurity (e.g., entropy or Gini index) is not above a certain threshold. This helps prevent splitting on features that don't significantly improve the quality of the splits.

After the tree is grown, it may be too complex or overfit to the training data, leading to poor generalization performance on new data. To address this, the **pruning process** is used to reduce the size of the tree by removing nodes that do not significantly improve the performance of the tree. The simplest method is to set a minimum number of samples per leaf, but more advanced methods such as cost complexity pruning or reduced error pruning can be used to balance between complexity and accuracy.

3. *Insight of Impurity Measures*
   We learned that the impurity measures are used during the learning process of decision trees. Here is quick overview of the two most common measures [13]:

   - **Gini index**
     Measures the probability of misclassifying a random sample from a set of samples. A Gini index of 0 indicates all elements of the set belong to the same class, while an index

of 1 indicates even distribution among all classes.
$Gini = 1 - \sum_{i=1}^{j} P(i)^2$
where j is the number of classes in the target variable (2 in our study) and P(i) the ratio of Pass over the total number of observations.

- **Entropy**
  Measures the uncertainty or disorder in a set of samples based on the distribution of class labels. A lower entropy indicates greater homogeneity, while a higher entropy indicates greater diversity.
  $E = -\sum_{i=1}^{n} p_i \log_2(p_i)$
  where n is the number of classes and p_i the probability of randomly selecting an example in class i.

2.2.3 Random Forest

Large decision trees, as mentioned above, tend to overfit on the training data, this limits the total gain in accuracy they can have while still being able to be put in production. In 1995 Tin Kam Ho published "Random decision forests" [1], the article proposed a new method of utilizing decision trees to allow them a continued gain of accuracy as they grow without suffering of overfitting. The method proposed by Ho is the splitting of the features across multiple trees. In 2001, Leo Breiman published the article "Random Forest" [3] combining his idea of Bootstrap Aggregation (Bagging) and Ho's idea of feature splitting (feature bagging). He then trademarked "Random Forests" in 2006.

Bootstrap Aggregation (Bagging for short) is the process of randomly sampling with replacement multiple subsets of the data. For each of these subsets we train our regression or classification algorithm on the data. Given we want to make B bags of data, with the training set (X, Y) with features X and corresponding responses Y, for $b \in [\![1...B]\!]$:

(a) Randomly and with replacement sample n elements from (X, Y) to create (X_b, Y_b).

(b) Train the decision tree f_b on the train data (X_b, Y_b).

The ensemble decision tree $\hat{f}$ defined by taking the set of decision trees we trained before $(f_b)_{b \in [\![1...B]\!]}$, can be defined as:

By the following equation in the case of a **Regression Problem**:

$$\hat{f}(x') = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x')$$

By taking the *majority vote* in the case of a **Classification Problem**.

Bagging helps dramatically reduce the variance of decision trees and as such, makes them more resistent to overfitting. Decision Trees using this technique are called bagged trees.

Feature Bagging is the process of selecting a subset of features for usage. During the growth phase of a standard decision tree, each node chooses the best feature to split using a greedy algorithm that minimizes the error. As such, bagged trees can have a very similar structure and in turn have high correlation in their predictions. To counter this problem,

the features that can be considered for splitting are randomly selected at each node. The standard training algorithm can then be applied at this node with these m features.

By combining these two methods, random forest algorithms are able to greatly reduce the variance of the predictions in comparaison to standard decision trees. This allows them to be less prone to overfitting.

"Random Forests do not overfit." - Leo Brieman

Random Forests algorithms can also be applied to **unsupervised learning problems**, to find a dissimilarity measure between data points in unlabeled data. The technique resides in generating artificial data and having the random forest classify if it is or isn't artificial.

2.2.4 Support Vector Classification (SVC)

Support Vector Classification (SVC) is a type of Support Vector Machines (SVM) that learns a decision boundary between different classes by maximizing the margin between the closest points of each class, which helps to improve the model's generalization ability [5]. Support Vector Machines are supervised classification machine learning models [4].

The decision boundary is defined by a hyperplane that separates the data points into different regions corresponding to each class. The distance between the hyperplane and the closest data points of each class is called the margin. SVC tries to maximize this margin while also minimizing the classification error. The data points closest to the hyperplane are called support vectors, and they play a critical role in defining the decision boundary.

The advantage of SVC is that it can handle non-linear decision boundaries by transforming the input features into a higher-dimensional space using a kernel function. In this higher-dimensional space, the data points become more separable, allowing SVC to find a decision boundary that can classify the data accurately.

2.2.5 Long Short Term Mermory (LSTM)

Sequences of data are hard to grasp using standard fully connected neural network model. To solve this problem Recurrent Neural Networks (RNN) were introduced, these networks read each node of data sequentially and process it with regards to their previous output. However, standard RNNs can only see the immediate past, this is illustrated by the gradient becoming very small as the sequences become longer (vanishing gradient).

To solve this issue Hochreiter and Schmidhuber introduced the Long Short Term Memory (LSTM) [2] in 1997. They introduced a memory cell that selectively stores or forgets information over time. The memory cell is controlled by three gating mechanisms, which regulate the flow of information into and out of the cell: the input gate, the output gate, and the forget gate. The input gate determines which information should be added to the cell, the forget gate determines which information should be discarded from the cell, and the output gate determines which information should be output from the cell.

Figure 3 shows the standard structure of an LSTM Cell, in a network, these cells would be put one after the other as the RNN goes through the input sequence. At the top, a bus takes the long term memory information from the previous cell state c^{t-1} to the current cell state c^t. To modify this memory, the network uses the previous input (short term memory) as well as the current input to make the cell forget some information, then add back some new information into the cell state. Once the memory has been updated the network uses the memory and the inputs to create the

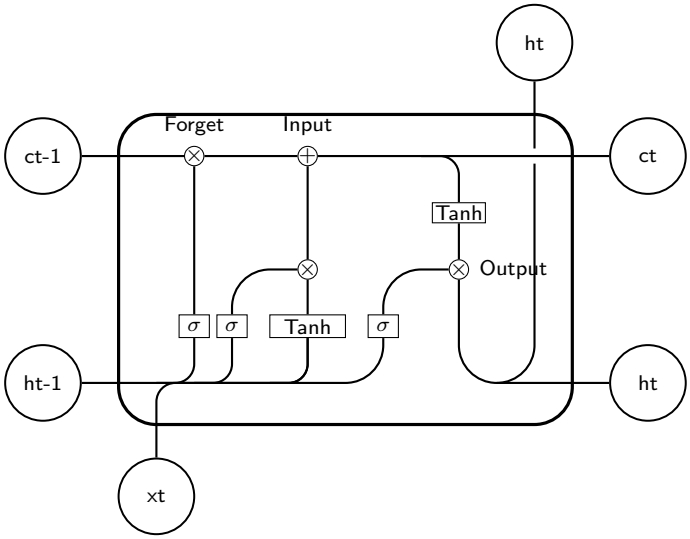


Figure 3: Standard LSTM cell architecture

new output h^t.

To apply the LSTM architecture to our classification problem, a dense layer is added to the ouput of the LSTM, this layer uses a softmax function to determine between the two classes.

2.2.6 Transformers

In 2017, Google Brain and Google Research published the article "Attention Is All You Need" which gave rise to the transformer [8]. The algorithm compares every element in the sequence with the others and gives importance/attention to certain relations. This allows the model to keep long term dependencies between elements, past and future.

Transformers rely on an encoder-decoder basis, the encoder gets the relations and creates a representation of the information. The decoder generates new information with it. First position information is added to elements/embeddings in the sequence, this is done using vectors that are generated by a predefined function in relation to the position of the element in the sequence. Then, the embeddings are passed through attention heads. An attention head separates an embedding into a Value (V), a Query (Q) and a Key (K), these are vectors. The Query and Key are used to weight the Value of elements against each other, for each element we sum the Value vector of every element in the sequence weighed against the element we are considering to create Z. This final sequence of vectors Z are the output of the attention head. By having multiple attention heads and combining their concatenated outputs with a weight matrix we are able to create a multi headed encoder. By using the representation the encoder gives us and feeding it into a recurent neural network, the decoder, we are able to translate texts, generate new texts amongst other things.

The Transformer architecture has achieved significant success in various natural language processing (NLP) tasks, including machine translation, text generation, and text classification. Based on it's architecture, many Large Language Models emerged. One of these would be Bidirectional Encoder Representations from Transformers [9] (BERT), a language model introduced by Google AI in 2018. It is pre-trained on a large corpus of text to generate high-quality language represen-

tations and has since been used as a basis for many other NLP models.

In the context of text classification, there exists the idea of Zero Shot text classification. The objective is to train the model to classify a subset of labels, and let it extrapolate to new labels. This allows the model to perform relatively well on new unseen labels, without having to label new data or having to retrain the model. We chose this approach due to the limited resources we had to train the models.
In Zero Shot classification, the model first converts the text and the labels into vector representations. These vectors reside in a latent space, which is meaningful only to the model. To classify the text, the model measures the distance between the vector representations of the text and the labels. The label that is closest to the text in the latent space is then assigned to the text as its label.

We used the distilBert and mobileBert models trained on the MNLI dataset [10] from hugging face. For the Zero Shot classification labels we used "*appropriate*" and "*inappropriate*".

2.3 Metrics

In this section, we will introduce and define the metrics that we will use to validate our model. Our problem is a binary classification problem, which means that we have only two outcomes: true or false. In our case, these outcomes correspond to inappropriate or not. Therefore, after making predictions, we will have four possible cases represented in Table 2.

Prediction	**Reality**	
	Positive	**Negative**
Positive	True Positive (TP)	True Negative (TN)
Negative	False Negative (FN)	False Positive (FP)

Table 2: Confusion Matrix

2.3.1 Accuracy

The first metric we will use is Accuracy [14]. This is defined as the number of correct predictions (true positives and true negatives) divided by all predictions:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

However, our dataset is unbalanced with 77% appropriate and 23% inappropriate language. This means that Accuracy alone is not enough to determine if our model accurately predicts inappropriate language. Therefore, we need to explore other metrics to validate our model.

2.3.2 Recall and Precision

To evaluate the performance of our model in identifying inappropriate language, we have also used precision and recall [14]. Precision is defined as the number of true positive predictions divided by the total number of positive predictions. In our case, this means that we are calculating the proportion of correct predictions for all inappropriate language identified by the model, including both true positives and false positives.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

However, as our main goal is to reduce the number of false negatives (i.e., cases where inappropriate language is not identified), we are also using recall as a metric. Recall is defined as the number of true positive predictions divided by the total number of actual positive elements. This means that we are calculating the proportion of correctly identified inappropriate language out of all the instances of inappropriate language present in the dataset, including both true positives and false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

By using both precision and recall, we can evaluate the overall performance of our model and determine whether it is effective in identifying inappropriate language. However, it's worth noting that optimizing for recall may increase the rate of false positives (i.e., cases where language is incorrectly identified as inappropriate). Therefore, we must strike a balance between recall and precision to achieve the best possible performance.

2.3.3 F-beta Score

While precision and recall provide valuable insights into the performance of our model, they do not take into account the relative importance of one metric over the other. In some cases, we may want to prioritize either precision or recall depending on the specific context of our problem.

To achieve a balance between precision and recall, we can use the F-beta score [11]. The F-beta score is a weighted harmonic mean of precision and recall, where the beta parameter determines the weight assigned to recall versus precision. When beta is set to 1, the F-beta score becomes the harmonic mean of precision and recall, also known as the F1 score:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}} \tag{4}$$

In our case, we want to prioritize recall over precision, as we want to minimize the number of false negatives, i.e., cases where inappropriate language is not identified. To achieve this, we can use the F2 score, where beta is set to 2. The F2 score gives more weight to recall than precision, which means that a model with a higher F2 score will have a higher recall than precision.

By using the F2 score, we can evaluate the performance of our model in a way that takes into account our priority of minimizing false negatives while balancing the trade-off with false positives.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN} \tag{5}$$

2.3.4 Inference Time

The last metric that we want to use to evaluate our models is the inference time. In the context of our problem, it's crucial to detect inappropriate language quickly in game chat so that the offending message can be removed before it reaches other players.

The inference time refers to the time it takes for the model to make a prediction on a new input. A fast inference time is important for our use case, as it allows us to detect inappropriate language in real-time and take immediate action.

With these five metrics, we can now more precisely evaluate the performance of our models and determine which ones are best suited to our needs.

3 Results

Models	**Metrics**				
	Accuracy	**Recall**	**Precision**	$\mathbf{F}_{\beta=2}$	**Inference** (avg ms per text)
Decision Tree	0.889	0.741	0.760	0.889	**2.5**
Decision Tree (GLoVE)	0.839	0.665	0.637	0.840	46.6
Random Forest	**0.916**	0.697	0.909	**0.914**	91.0
Random Forest (GLoVE)	0.903	0.619	**0.926**	0.899	326.2
Support Vector Classifier	0.889	0.598	0.870	0.885	18949.8
LSTM	0.912	0.767	0.828	0.911	692.9
LSTM (GLoVE)	0.892	**0.803**	0.692	0.891	637.4
mobileBERT (MNLI)	0.575	0.563	0.621	0.574	NA

Table 3: Models benchmark

As you may notice on Table 3, all of these metrics evaluate the models on the dataset, which is not the final goal of our project. Indeed we want the models to perform well on "In-Game Chats", even-though the training/testing data is Social media Comments.
In extension of this study, we therefore need to build a small labelled dataset of In-Game Chats to test all the models on, or even fine tune models such as BERT. Unfortunately, such data are yet not openly accessible.

Due to time restrictions, the mobileBERT model couldn't be run on the entire data, as such the score are only predicted on 10000 samples of the test data.

4 Final Model

The goal of this study is to find a machine learning model robust enough to classify inappropriate content reliably enough to be moved into production and light enough to be able to run on a game server. In this optic, we wanted a model with a low inference time and with a high F2 score. This is why we chose the Decision Tree paired with a count vectorizer: the inference time is low, the F2 score (0.889) is close to the the best score (0.914) and all other metrics are balanced.
However, we consider the fact that we might have to come back on our decision. That is because the model could see different type and format of texts depending on the game chat the model monitors. With this in mind, we propose the use of the Random Forest with GLoVE using an average of the GLoVE embeddings, if the decision tree fails. This is because the GLoVE embeddings will allow it to use a very large vocabulary while the random forest should be able to adapt better to this new vocabulary.

4.1 Motivations

The production model needs to run in real-time on top of a game chat system to be able to constantly monitor and suppress unfriendly behaviour in the chat. The model would have had to run essentially on a cpu as it is what most small game servers use, and it needs to be fast as to not incur a noticeable delay in the chat system. The model should also be accurate enough not to cancel appropriate languages and remove inappropriate chat messages.

4.2 Usage

1. Download the pretrained tokenizer and model (from the *final_model* folder).
2. We advise you to use Anaconda. You can configure the environment using the *requirements.txt*.
3. Follow the instructions in *final_model.ipynb*.

References

[1] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, 278–282 vol.1. DOI: 10.1109/ICDAR.1995.598994.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] L. Breiman, *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/a:1010933404324. [Online]. Available: https://doi.org/10.1023/a:1010933404324.

[4] R. G. Brereton and G. R. Lloyd, "Support vector machines for classification and regression," *Analyst*, vol. 135, pp. 230–267, 2 2010. DOI: 10.1039/B918972F. [Online]. Available: http://dx.doi.org/10.1039/B918972F.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162.

[7] P. Gupta, *Decision trees in machine learning*, Nov. 2017. [Online]. Available: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052.

[8] A. Vaswani, N. M. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *ArXiv*, vol. abs/1706.03762, 2017.

[9] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[10] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. [Online]. Available: http://aclweb.org/anthology/N18-1101.

[11] J. Brownlee, *A gentle introduction to the fbeta-measure for machine learning*, Jan. 2020. [Online]. Available: https://machinelearningmastery.com/fbeta-measure-for-machine-learning/.

[12] B. Jijo and A. Mohsin Abdulazeez, "Classification based on decision tree algorithm for machine learning," *Journal of Applied Science and Technology Trends*, vol. 2, pp. 20–28, Jan. 2021. [Online]. Available: https://www.researchgate.net/publication/350386944.

[13] S. Dash, *Decision trees explained-entropy, information gain, gini index, ccp pruning.*. Nov. 2022. [Online]. Available: https://towardsdatascience.com/decision-trees-explained-entropy-information-gain-gini-index-ccp-pruning-4d78070db36c.

[14] S. Harispe, *Introduction to machine learning*, Feb. 2023.