Increasing Immersion Through Emotional Response Natural Language Processing

Jackson Ingraham, Jackson Westbrook

What problem are you addressing?

Talking to a video game non-playable character (NPC) limits the creative freedom of a player as they are restricted to predetermined decisions, which can negatively impact the experience. When making an NPC, a developer has to manually create each interaction that a player can make.

For example, in the game *Stardew Valley* players are given an option of three unique responses when talking to an NPC. Each option has a pre-programmed response from the NPC, and a reward or punishment for the player. The problem is not the limited responses from the NPC, but instead the limitation set upon the player that constricts their creative freedom. Having a developer create an option for every response a player may think of is unreasonable. This can be an issue as the player creates their character, and thus should have full control over their personality and actions. This is not a problem in games like *The Witcher 3* as the character is already predefined.

As another example, the role-playing game Baldur's Gate 3 has the player create their own persona that goes through a story. This story, while not strictly predetermined, only has so many options available, as each option must be programmed in by the developers. The game feels amazing to play, with one caveat. Sometimes, the player may be given four options that they do not feel fit their character. It is easy to just pick a random answer and move on, but a game should strive to give the player the best experience possible.

If we allow the player to say whatever they want, then they respond with higher scores when asked a series of questions relating to the feelings derived from playing the game they are playing because they are given the opportunity to be themselves in a way that a developer could not predict.

Why is this problem important?

A major hurdle in getting immersed in a world is when the user feels their character acts differently from what they expect. Story-based games get around this by giving the user control of a predetermined character, one with their own motivations and goals with the intent that the user connects with them at some level. Utilizing NLP, this level of immersion can be extended to games outside of this genre by enabling users to react in a manner that impacts the characters and the world around them in an appropriate manner.

Its use case still applies outside of this context as well, serving as an effective method for data collection on specific products/topics to determine how people generally feel about it.

The hypothesis is that if we introduce the ability for a player to say anything that they want by using a sentiment analysis algorithm, then they will feel more connected to the gameplay experience compared to basic sentiment analysis when asked to respond to a list of statements.

Relevant Literature

[1] provides a general overview of stance detection, aiming to aggregate methods, evaluations, and papers relevant to each methodology provided. It does not directly address the performance, but rather the overarching design and theory behind stance detection along with traditional and LLM alternatives. It additionally provides datasets ranging from 2017-2024 categorized by type for training stance detection models. Desired datasets are included in the footnotes of the hyperlinked papers.

[2] provides insight on general features to use in a dataset catered towards stance detection of tweets. Some of these features include original topic, author, id, and pre-processed text. It also provides background information on how the data was gathered.

How will you address this problem?

We will address the problem by using sentiment analysis to create an algorithm that can be used by video game developers to create more interactive NPC interactions without hard coding each individual character's reactions to player inputs. This will also allow the players a wider range of possible inputs, as they will be able to type in responses instead of choosing from a predefined list. The player will be prompted by dialogue from an NPC, and be given a command line-like typing box where they can freely type what they want their character to say. For this project, we will be using this to create a very simple shopkeeper that a player can buy things from. The main thing that we are creating, however, is the model used to interpret the interactions with the NPC.

Our primary methodology is to utilize a stance detection algorithm to augment traditional sentiment analysis by highlighting the sentiment of one specific keyword. Our experimental scenario limits the inputs to two sides: the user and the developer. The user is expected to provide some text after being prompted by the developer while the developer queries for the semantics of a specific keyword. This is accomplished by including two features alongside the base text being analyzed as is usually the case for sentiment analysis: the keyword and whether or not the base text included the keyword which is denoted as 'seen'. The separation of the keyword and the 'seen' feature is included so as to not lose the relation once both the base text and keyword are encoded. [2] suggests additional features such as author or original topic, but these features are not accessible in our experimental set up and thus left out.

To determine the effectiveness of the keyword feature and the seen feature on classifying the sentiments, all permutations of the inclusion of features iterated through. Despite having an

impact on all classification methods, the comparison was limited to Random Forest, Naive Bayes, and Support Vector classification methods due to the speed at which the methods converged relative to methods which required a neural network.

All features are then used to form neural network models used for classification: one dense neural network and one which utilizes a pre-trained BERT model. Due to time constraints, these models were only trained on all features. The models would be compared against themselves for performance alongside the models prior, with the models utilizing no additional features serving as a baseline for just sentiment analysis.

What are some alternatives and how do you justify your approach?

Current alternatives include static, pre-programmed dialogue options or branching narrative systems. These methods restrict player creativity and limit immersion and only effectively function when the character is also used as an active medium for storytelling by the creator rather than a way for users to interact with the world. In the event where the user is expected to create their own character and motivations, the narrative formed by creators often forms a disconnect between the user and the story when the expectations on how the user is expected to act conflicts with reality.

Our approach, utilizing NLP and emotional analysis, allows for greater freedom while maintaining realistic and context-appropriate responses from NPCs. The NLP serves as a dynamic user interface which transforms the input into a context which is useful to the developer. The scope is limited primarily to sentiment and stance analysis with fixed responses to allow for a focus on the NLP and remain feasible within the timeframe of the project. Extending this interface would enable the use of LLMs and more complex outputs for more dynamic conversations between users and NPCS, which is the primary motivation.

We came across some challenges regarding the output of the model. Different models included different values outputted for the sentiment analysis. For example, one would output negative one for a negative sentiment, a zero for neutral sentiment, and a one for positive sentiment, while another may swap the neutral and positive sentiment output values so that a zero means it is positive, while a one would be neutral sentiment.

Training the dense model also came with its own set of challenges. Failure in the model's ability to learn in early epochs could be attributed to both flaws in the model architecture and hyperparameter fine-tuning, requiring both to be reconfigured many times throughout the training process. Early stopping was not a method that could be used on the validation data accuracy, as both the training and validation accuracy would remain constant throughout much of the training. Early iterations of the model suffered heavily from overfitting, a problem swiftly solved with the inclusion of regularization techniques. After the inclusion of regularization, the model stopped learning entirely for a decent period of time due to convolutional layers abstracting away the initial state too heavily. This problem persisted until the model was adjusted to consist solely of dense layers, where the model would once again start learning.

Another challenge we faced was the actual feel of the gameplay. The couple of playtesters I got to test the shopkeeper decided that while they could feel connected to the character, typing their responses out was not something that they enjoyed.

How will you evaluate your approach?

We began our training on Target-Specific and Multi-Target Datasets found in [1], as the stance and sentiment detection portions need to function competently before fine-tuning for our specific use case. The dataset itself is split in a 80-20 train-validation split to ensure that the models do not overfit on the training data and allows for measures such as early stopping. Models produced 2-3 logits depending on the model type. Random Forest, Support Vector, and Naive Bayes used 2 logits with neutral defined as the logits being equal, as these models were capable of having all zero logits leaving the classification problem as undefined. Other models used 3 logits for negative, neutral, and positive sentiments respectively. Since the data is categorical in nature, CrossEntropyLoss is used as the loss function where not already decided by the Pytorch or Sklearn.

All methods utilize accuracy as their main metric, as false positives and false negatives hold the same impact to the user regardless of the scenario. Comparing the performance of all models also introduced the additional metric of runtime of the model when evaluating, as the environment favors a faster response for the player.

Finally, we tested the effectiveness of the model in making the player feel connected to the character they are playing. We told three playtesters a list of two statements, and requested that they respond with a value between 0, strongly disagree, and 5, strongly agree. The statements were as follows: “Being able to type out your answer makes you feel more connected to the character you are playing,” and “You prefer being able to type out your answers compared to preselected choices.”

What are your main findings?

Initial testing involved determining the impact that the new features had on the overall accuracy of the models. All the models utilized Bert tokenizer for both the keyword and the actual text, with the ‘seen’ feature being a boolean. Preprocessing was left to the Bert tokenizer to allow for the most accurate vector representation since the tokenizer expects un-processed inputs. The tensor was then flattened and fed into each of the corresponding models, outputting two logits with an undefined state being neutral. The outcome of this is shown in *Table 1*.

Validation Accuracy	Random Forest	Naive Bayes	Support Vector
Keyword + Seen	40.68	28.57	30.27

Keyword	42.13	28.57	30.27
Seen	26.63	25.18	27.36
None	27.36	25.18	27.36

Table 1: Performance of models with label permutations

The inclusion of the keyword feature substantially increases the performance of all the models, indicating the feature is useful in deriving information about the semantic of the keyword in the sentence. In contrast, the seen keyword actively hinders the models and their ability to classify sentiments by either remaining equal in accuracy or lessening the model accuracy in the case of Random Forest. Since adding the seen parameter actively hinders the model from becoming more accurate, it indicates that the model either suffering from more overfitting in training or resulting in the model forcing itself to fit around more specific boundaries causing a lower validation accuracy.

Table 2 shows the validation accuracy and runtimes of all the models when trained using all the optional features provided in *Table 1*. The same tokenizer was used for all models aside from transfer learning, which required a special AutoTokenizer to fit the data to the transformer. The dense model itself underwent several iterations, originally starting off as a set of convolutional layers which fed into the linear layers. Once the inclusion of regularization was added using layer norm, dropout, gradient accumulation scheduling, stochastic weight averaging, gradient clipping, and weight decay, the convolutional layers actively inhibited the model from learning. Instead, these layers were removed in favor of additional dense layers which doubled in size several times before shrinking. LeakyReLU is used as the activation function for all the layers to prevent any vanishing gradients. The dense layer trained for 50 epochs before terminating to keep the training time comparable to that of the transfer learning model.

	Random Forest	Naive Bayes	Support Vector	Dense	Transfer Learning
Runtime (ms)	160	118	124	242	4417
Validation Accuracy	40.68	28.57	30.27	55.69	74.09

Table 2: Performance of models with all labels

Runtime directly correlates to validation accuracy, likely due to higher complexity models requiring longer runtimes to evaluate a single instance. All models outperformed the baseline models, but only random forest, dense, and transfer learning are viable as stance detection models as they are the only ones outpacing the ⅓ chance of guessing the correct answer. Random forest is also the lowest of the 3 models, despite having a comparable runtime to the dense model while also unable to improve without additional features like the transfer

learning model. This ultimately leaves dense and transfer learning as the two viable options for stance detection, with dense being preferable if you have time prior to launch to tune hyperparameters manually as it has potential to evaluate quicker while transfer learning is quicker by requiring little tuning at all.

Experimentally, the transfer learning model specifically fails at identifying statements where the keyword is neutral, producing logits for similar values for both positive and negative sentiments and completely neglects utilizing the neutral sentiment logit whenever any semblance of non-neutral sentiment exists in the sentence. Checking if the difference between positive and negative sentiment logits is significant enough is a theorized way to improve the model. The dense model suffers from a similar problem with classification, ultimately relying on collapsing back down to sentiment analysis independent of the keyword. Further testing is required to determine whether or not additional fitting would result in a stronger reliance on the keyword feature or just result in the model overfitting on the training data.

The playtesters' scores are in the chart below. It was determined that while players may feel more connected to the character, which was still scoring lower than expected, the players did not prefer this method of communication in a game. Complaints included the fact that they did not enjoy the time it takes to type in responses, compared to just clicking an option.

	Connected	Preferrence	Average
Player 1	3	1	2
Player 2	2	2	2
Player 3	4	1	2.5
Average	3	1.33	2.17

Table 3: Playtesters opinions charted

Real world interactions

Practical usage for our work includes gathering sentiment from a specific topic on social media or reviews. Companies and individuals can gather intel on how individuals feel about a specific product by scrapping posts, determining their relevance by identifying the sentiment of the keyword in a post, and sorting by what consumers like or dislike about the product. The number of posts of a given sentiment can be used to indicate how the general public feels about a specific product for market research purposes.

Arguably, such a tool could be used with the intention of silencing the populace by ensuring posts which correspond to a specific keyword only contain a specific type of sentiment. Since every single possible unintended interaction cannot be accounted for, the best way to combat such an occurrence would be by restricting the model capabilities to only specific

keywords, building it up over time as demand in the model grows. This would inhibit how quickly the model is capable of growing, but would ultimately be the best way to control the use case of it.

Limitations

A major limitation stems from the fact that despite our primary reasoning that players would appreciate the free agency of being able to freely interact with the world how they chose, the model is still limited in what the developers intend. The developers still have to expect to parse for specific keywords, and have a response planned for every route or combination of emotions they check for. It still provides the degree of flexibility for the side of the player, but completely neglects that flexibility for the developer. Additionally, the developer cannot account for absolutely everything a player could say. Even in a test question like "Would you like an apple?", the player could throw out a completely random string unrelated to the question that the developer is unable to account for. This can slightly be appeased by mixing stance detection with sentiment analysis to determine the overall tone of the text mixed with the tone of a specific keyword to attempt to provide the developer with more flexibility, but it still cannot account for every scenario.

The base methodology in testing the features itself suffers from a lack of testing on neural models like the dense network and transfer learning. Since the mathematical models utilize more of a 'hard boundary' from features where a specific border is defined on a higher dimensional plane, features which produce conflicting information can result in a less stable boundary as shown with the 'seen?' feature decreasing the validation accuracy when included in the training. To ensure that this had a similar effect on a neural network, the feature should have been compared on one. Although neural networks can shut down features they ultimately deem useless to classification through weights, it needlessly imposes additional work on the network and hinders the network's ability to classify.

Findings on the dense neural network were greatly hindered by constantly redesigning the architecture and fine-tuning the hyperparameters, to the point that an optimal combination of both was unable to be found in a timely manner. Since the model was trained without using a GPU, the model took hours to run before it could be determined whether or not a change was effective. Additionally adding layers to give the network more parameters to work with would only increase this runtime, making it more difficult to train. Better hardware would have expedited this process significantly allowing for more frequent fine-tuning and better optimization overall. Findings were also inhibited by the dataset. Only 2062 entries were in the base dataset, yet there was not enough time to augment other datasets found in [1] to include the same features. K-splits would have been a valid way around this, had the model not been so time-consuming to train on its own. Stratification was used to help eliminate some of the burden on variations between the validation and training datasets, but still remained not ideal for our use case. Findings would have also benefited from being compared to general sentiment analysis,

since they overlap heavily and would indicate whether or not the findings collapse back into sentiment analysis outside of testing.

Application Prerequisites

For restrictions in applications as the program is currently implemented, all that would be required is a desired keyword(s) to check for coupled with some source of text, be it an arbitrary front end or parsing some social media site. The model for both the dense and transfer learning (which would likely be preferred by the user due to their higher performance) was too large to be pushed to the Github, but can easily be reconstructed through the notebook by the end user by running the code blocks and using the training dataset included in the Github, assuming that the model is not just directly shared via a third party file sharing service such as Google Drive.

Once the desired model is recovered, the *model.py* file serves as an effective interface between the text/keyword and the models. It allows the end user to select their model and pass the text/keyword into a method which handles the pre-processing, feature extraction, and formatting for the model input while producing a one-hot model output for the end-user to utilize in whatever logic they so chose. Modifications to this would be required, specifically to the run model method, in the event that the priority is to parse as many keyword/text permutations as possible as the current set-up does not support batching since that is outside of its intended use case. Otherwise, the model is fully capable of running independently of the front-end.

The front-end itself would need to be completely re-created to be customized to the use case of the end user. The current front-end is meant to serve as a bare-bones demo for the capabilities of the model rather than be the only way the model can be used. Social media parsing programs can largely scrap the demo entirely, loading the data from a file and passing it into the method. Alternatively, the data could be loaded directly into the model class found in *model.py* and pre-processed all at once, optimizing that stage of the step and ensuring that the data is directly passed into the model without any other interference.

Even in the event that the model is meant to be used within a game environment, the user heavily benefits from re-writing the front-end to suit their own design due to how easy it is to connect their text to the interface. While the PyGame window has potential for reuse if the same GUI is desirable by the user, the questions, keywords, and logic with the resulting sentiments would need to be reworked to produce the outcome intended by the user. Since the outcome is something inherently unique per dialogue option, chaining together specific reactions or questions would not be challenging nor would switching around the pre-existing text and keywords since they are isolated from the rest of the code.

While python libraries are largely what the code would be restricted to use due to needing to activate the neural network, a game would largely benefit from attempting to add graphics to the interface at some point to further increase the immersion of the experience. This would work in conjunction with the model to further improve emotional response by the player to make them

actually feel like they are a part of a world where the things they say and do actually impact the environment around them.

References

[1] Bowen Zhang, , Genan Dai, Fuqiang Niu, Nan Yin, Xiaomao Fan, Hu Huang. "*A Survey of Stance Detection on Social Media: New Directions and Perspectives*." (2024). https://arxiv.org/pdf/2409.15690

[2] Allaway, Emily, and Kathleen McKeown. “Zero-shot Stance Detection: A dataset and model using generalized topic representations.” *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, https://doi.org/10.18653/v1/2020.emnlp-main.717.