Emotion Classification Report – Technical Log

Preprocessing and Initial Setup

Hyperparameters Initialized

Set max_features=500 to reduce dimensionality and focus on the most frequent terms. Chose stop_words='english' to remove common but less informative words, aiming for a cleaner feature set. Initially, n_jobs=1 was selected for simplicity, though future optimizations may include parallel processing to decrease training time.

Dataset Dimensionality Considered:

The initial feature space was vast; thus, limiting features and removing stop words were crucial steps to manage computational load and focus on meaningful text analysis.

Navie Bayes – Hyperparameter Selection

Vectorization Hyperparameter Selection

vectorizer__max_features: This parameter limits the number of words (features) to consider based on their frequency across the corpus. The options [None, 500, 1000, 2000] offer a spectrum from no limit to increasingly restrictive limits. Not setting a limit (None) means all unique words are considered, which might capture more detail but also increase noise and computational cost. On the other hand, limiting the features can focus the model on the most relevant information but might risk losing contextual nuances. The choice of limits (500 to 2000) aims to find a sweet spot where the model remains computationally efficient without significant loss of accuracy.

vectorizer__ngram_range: N-grams are combinations of adjacent words in the text, with (1, 1) representing unigrams (single words), (1, 2) including both unigrams and bigrams, and (1, 3) extending up to trigrams. This parameter helps capture context and phrase meaning that might be missed when only considering single words. The range options allow the model to learn from simple to more complex textual patterns, potentially improving its understanding of the data.

vectorizer__stop_words: Stop words are commonly used words (such as “the”, “a”, “in”) that are often removed before processing text data. The options [None, 'english'] compare the effect of excluding these words against keeping them. Removing stop words can reduce the dataset's dimensionality and focus the model on more meaningful words, improving performance. However, in some contexts, stop words might carry important meaning or nuance, so their removal is tested empirically.

Classifier Hyperparameter Selection

classifier__alpha: This parameter controls the smoothing applied to word frequencies to manage the issue of zero probabilities in the Naïve Bayes algorithm. The values [0.01, 0.1, 1.0, 10.0, 100.0] explore a range from very little smoothing to very high smoothing. Low values of alpha make the model more sensitive to the training data, which can improve accuracy if the training data is representative but may lead to overfitting. High values decrease sensitivity to the training data, reducing overfitting risk but possibly underfitting the model. The goal is to find an optimal level of smoothing that balances bias and variance.

classifier__fit_prior: This boolean parameter decides whether to learn class prior probabilities (True) or to use a uniform prior (False). Learning the prior can be advantageous if the training set class distribution is reflective of the real-world scenario, as it allows the model to adjust its predictions based on the overall distribution of classes. Conversely, not fitting the prior can be useful if the class distribution in the training set is skewed or not representative of actual conditions.

Logistic Regression – Hyperparameter selection

TF-IDF Vectorization Hyperparameter Selection

Feature Count: The Feature Count hyperparameter directly controls the number of terms to keep based on the TF-IDF score across the corpus. The selected values [100, 500, 1000, 1500, 2000] offer a wide range to examine the model's performance across different levels of information granularity. A smaller number of features can make the model faster and less prone to overfitting but may miss important terms that could improve classification accuracy. Increasing the number of features can capture more detailed patterns in the text but also increases the risk of overfitting and computational complexity. The aim is to find an optimal balance that maximizes predictive performance without unnecessarily burdening the model.

Binary: The Binary parameter, with options [True, False], determines whether to simply use the presence or absence of terms as features (True) or to use their TF-IDF scores (False). When set to True, the model ignores term frequency in the document and focuses solely on whether terms appear or not, potentially reducing the impact of very frequent terms. This can be useful in situations where the occurrence of terms is more important than their frequency. When False, the model considers the actual TF-IDF scores, which can help in emphasizing terms that are important and unique to specific documents.

Use_IDF: The Use_IDF parameter decides whether to weigh terms by their inverse document frequency (True) or not (False). Applying IDF scaling (True) diminishes the weight of terms that appear frequently across documents, thereby focusing on terms that provide more unique information about a document's content. This is generally beneficial for classification as it helps the model to prioritize distinctive terms over common terms. Without IDF scaling (False), the model relies purely on term frequency within documents, which can be beneficial if the raw frequency of terms is more indicative of the document's classification.

Logistic Regression Hyperparameter Selection

Max_Iter: The Max_Iter hyperparameter sets the maximum number of iterations for the logistic regression algorithm to converge to a solution. The selected values [100, 500, 1000, 2000, 3000, 4000] provide a broad spectrum to assess how quickly the algorithm converges and at what point additional iterations no longer yield significant improvements in model accuracy. Starting with a lower number of iterations can be computationally efficient but might result in the model not fully converging, especially if the optimization problem is complex. Increasing the number of iterations allows the model more opportunity to converge but at the risk of higher computational cost and potentially diminishing returns on model performance improvement.

Decision Tree – Hyperparameter selection

decisiontree__max_depth: This parameter controls the maximum depth of the tree. The depths [10, 50, 100, 200] represent a range from shallow to very deep trees. A shallow tree (max_depth=10) might be too simple to capture the underlying patterns in the data, leading to underfitting. Conversely, a very deep tree (max_depth=200) can capture very fine details of the training data, potentially leading to overfitting by memorizing the data rather than learning the general patterns. The selection aims to find an optimal tree depth that balances complexity with the ability to generalize.

decisiontree__min_samples_split: This parameter determines the minimum number of samples required to split an internal node. The values [2, 50, 100] range from allowing splits on very small groups of samples to requiring larger groups for a split. A lower value means that the tree can be more complex and possibly overfit, while a higher value encourages the tree to be more generalized by forcing splits to only occur at nodes with a substantial number of samples, thereby reducing model complexity.

decisiontree__min_samples_leaf: Similar to min_samples_split, min_samples_leaf defines the minimum number of samples a leaf node must have. The options [1, 20, 50] explore the impact of allowing leaves with very few samples versus requiring more substantial evidence to form a leaf. Setting this parameter to a higher number helps in smoothing the model, especially in regression

tasks, by avoiding leaves with very few samples which can result in models that capture noise in the training data.

decisiontree__class_weight: This parameter is used to weigh the importance of each class based on the values [None, 'balanced']. For imbalanced datasets, setting this to 'balanced' can help by adjusting weights inversely proportional to class frequencies in the input data, thus giving more weight to underrepresented classes. This can improve the model's performance on minority classes, which might be critical in applications where all classes are of equal importance.

Random Forest – Hyperparameter selection

rf__n_estimators: This parameter determines the number of trees in the forest, with options [100, 200, 300]. More trees generally increase the model's robustness and accuracy, as the ensemble can better capture the variability in the data through averaging or majority voting. However, adding more trees also increases computational cost and time. The selected values aim to find a balance between performance and efficiency, with a moderate to high number of trees to ensure adequate model complexity without excessive computational demand.

rf__max_depth: Similar to the decision tree model, max_depth controls the depth of each tree in the forest. The depths [10, 50, 100, 200] allow for exploration from relatively shallow to very deep trees. Shallow trees (lower max_depth) make the model faster and less prone to overfitting but might not capture complex patterns. Deeper trees (higher max_depth) can model more complex relationships but risk overfitting. This parameter helps balance the model's bias and variance by limiting how deep the trees can grow.

rf__min_samples_split: This parameter sets the minimum number of samples required to split an internal node, with options [10, 50, 100]. Higher values prevent the creation of nodes that split too few samples, reducing model complexity and helping prevent overfitting. It encourages more generalization by requiring a significant number of samples to justify further partitioning within the trees.

rf__min_samples_leaf: It specifies the minimum number of samples a leaf node must have, with values [10, 20, 50]. By setting a minimum threshold, it ensures that leaf nodes have a sufficiently large number of samples, which contributes to smoothing the prediction and reducing overfitting. This parameter is particularly useful for creating more stable and generalized models by avoiding overly specific or noisy leaf nodes.

rf__max_features: The max_features parameter determines the number of features to consider when looking for the best split, with options ['sqrt', 'log2']. sqrt (square root of the total number of features) and log2 (logarithm base 2 of the total number of features) are heuristic methods to limit the feature space, making each split decision more robust and diverse across trees. Reducing the number of features considered for each split can decrease overfitting risk and improve generalization, as each tree in the forest will likely focus on different subsets of features, increasing ensemble diversity.

Catboost – Hyperparameter selection

catboost__iterations: This parameter specifies the number of trees (or the number of boosting stages) in the model, with options [100, 500, 1000, 2000]. Increasing the number of iterations allows the model to become more complex by adding more trees, which can improve accuracy up to a point. However, beyond a certain number of trees, gains in accuracy diminish, and the model becomes more prone to overfitting and increases in training time. The chosen range allows for exploring the impact of model complexity on performance, from relatively quick and simple models to more complex and potentially more accurate ones.

catboost__learning_rate: The learning rate controls the step size at each iteration while moving toward a minimum of the loss function, with values [0.01, 0.05, 0.1, 0.2]. A smaller learning rate makes the model training more gradual and thorough, potentially leading to better generalization but requiring more iterations (trees) for the model to converge. A higher learning rate speeds up training but can overshoot the minimum loss, leading to suboptimal solutions and overfitting. The specified range tests the balance between training speed and model accuracy, aiming to find an optimal rate that achieves high performance without overfitting.

catboost__depth: This parameter determines the depth of each tree, with options [6, 10, 20, 30]. Tree depth is a critical factor in model complexity; deeper trees can model more complex patterns and interactions but also increase the risk of overfitting by capturing noise in the training data. Conversely, shallower trees are faster to train and less prone to overfitting but might miss capturing more complex relationships. The range from 6 to 30 allows for assessing the impact of depth on model performance, from moderate to very deep trees, to identify the depth that offers the best trade-off between complexity and generalization.