1. **Methodology Overview**:

The code implements a text classification pipeline using PySpark, a powerful framework for large-scale data processing. The goal is to build a model that can classify the sentiment of movie reviews as either positive, negative, or neutral.

1. **Architecture and Logic**:
   * The code starts by importing necessary modules from the PySpark library.
   * It defines a list of special characters to be removed from the text data.
   * The pipeline consists of several stages:
     + **Tokenization**: The input text is split into individual words or tokens.
     + **Stopwords Removal**: Common stopwords and special characters are removed from the tokenized text.
     + **Word Vectorization (Word2Vec)**: Words are converted into numerical vectors. This step captures semantic similarities between words.
     + **Label Indexing**: Sentiment labels (e.g., 'positive', 'negative', 'neutral') are converted into numerical indices for model training.
     + **Random Forest Classifier**: A Random Forest model is trained on the vectorized text data to classify sentiments.
     + **Label Conversion**: Predicted numerical indices are converted back to sentiment labels for easier interpretation.
   * The pipeline is assembled with these stages.
   * The model is trained on the training data (**spark\_df\_train**) using the **fit()** method.
   * Predictions are made on the test data (**spark\_df\_test**) using the trained model.
   * The accuracy of the model is evaluated using a multiclass classification evaluator.
   * Finally, sample text data is prepared and predictions are made on it using the trained model.
2. **Key Functions/Modules**:
   * **Tokenizer**: Splits text into individual words or tokens.
   * **StopWordsRemover**: Removes common stopwords and specified special characters from text.
   * **Word2Vec**: Converts words into dense numerical vectors.
   * **StringIndexer**: Converts categorical labels into numerical indices.
   * **RandomForestClassifier**: Trains a Random Forest model for classification.
   * **MulticlassClassificationEvaluator**: Evaluates the accuracy of the classification model.
3. **Algorithms/Techniques**:
   * **Word Embeddings (Word2Vec)**: It's a technique used to represent words in vector space where words with similar meanings have similar vectors. It captures semantic relationships between words.
   * **Random Forest Classifier**: It's an ensemble learning method that builds multiple decision trees during training and outputs the class that is the mode of the classes (classification) of the individual trees. It's robust, scalable, and suitable for text classification tasks.

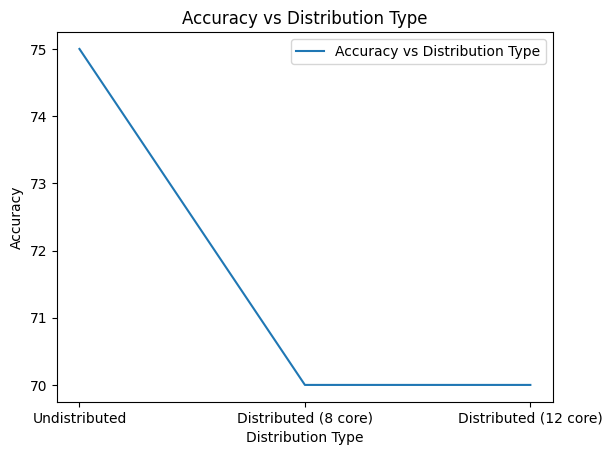
Overall, the code demonstrates the process of building and evaluating a text classification model using PySpark, incorporating text preprocessing, feature engineering, model training, and evaluation steps.

**RESULTS: -**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **UNDISTRIBUTED** | **DISTRIBUTED**  **(8 Cores)** | **DISTRIBUTED**  **(12 Cores)** |
| **TIME** | **480 sec** | **390 sec** | **270 sec** |
| **ACCURACY** | **75%** | **70%** | **70%** |
| **EMBEDDING SIZE** | **30** | **30** | **30** |
| **RANDOM FOREST**  **(trees)** | **100** | **100** | **100** |
| **Depth** | **4** | **4** | **4** |
|  |  |  |  |

**A graph of different sizes of blue rectangular objects

Description automatically generated with medium confidence**

****

**CONCLUSION**

**Conclusion:**

This project has been invaluable in enhancing our skills, particularly in leveraging PySpark for distributed computing. While we acknowledge that the accuracy of the RandomForest classifier in PySpark was not optimal, we believe that increasing the embedding size could potentially improve accuracy. Nonetheless, the notable improvement in processing time achieved through distributed computing was remarkable and underscored the efficacy of utilizing PySpark for large-scale text processing tasks.

**Future Work:**

Moving forward, we plan to explore various techniques to enhance accuracy, such as incorporating stemming and lemmatization. Additionally, we aim to scale our system by increasing the number of systems in our cluster, which could further optimize processing time. Furthermore, we intend to experiment with larger embedding sizes to strike a balance between processing time and accuracy, thereby advancing the performance of our text classification model.