***Intensity Analysis Project Proposal***

**Objective:**

The goal of this project is to develop an intelligent system using Natural Language Processing (NLP) to predict the intensity of emotions (happiness, anger, sadness) in text reviews. This predictive capability will enable businesses to proactively optimize their processes and improve overall customer satisfaction.

**Focus Areas:**

* Data Collection: Gather text data with labeled intensity information.
* Data Preprocessing: Clean and preprocess the text data for machine learning.
* Feature Engineering: Extract and select features relevant to intensity prediction.
* Model Selection: Choose suitable machine learning algorithms for classification.
* Model Training: Train the models using the preprocessed data.
* Model Evaluation: Evaluate the model's performance using appropriate metrics.
* Hyperparameter Tuning: Optimize hyperparameters to enhance model accuracy.
* Deployment: Deploy the trained model for real-time predictions.

**Recommended Steps:**

* Data Collection:
  + Collect text data with intensity labels from various sources such as social media, customer reviews, and feedback forms.
  + Use APIs like Twitter API, web scraping, or public datasets like IMDB, Yelp, or Amazon reviews.
* Data Preprocessing:
  + Remove noise (e.g., HTML tags, special characters).
  + Tokenize text into words.
  + Normalize text (e.g., lowercasing, stemming, lemmatization).
  + Handle missing values.
* Feature Engineering:
  + Extract features such as n-grams, TF-IDF scores, and word embeddings (e.g., Word2Vec, GloVe).
  + Identify and select key features impacting intensity prediction.
* Model Development:
  + Use machine learning techniques like logistic regression, SVM, or deep learning models like LSTM or BERT.
  + Train the models on the preprocessed data.
  + Fine-tune models for improved accuracy.
* Testing and Validation:
  + Split data into training and testing sets.
  + Evaluate model performance using metrics like accuracy, precision, recall, and F1-score.
  + Address any issues identified during testing.
* Hyperparameter Tuning:
  + Use techniques like Grid Search or Random Search to find the best hyperparameters.
  + Optimize the model for higher accuracy.
* Deployment:
  + Deploy the trained model using tools like Flask or FastAPI for real-time predictions.
  + Ensure the deployment environment supports scalability and reliability.

**Timeline:**

* Week 1:
  1. Data Collection and Preprocessing.
  2. Feature Engineering and Selection.
* Week 2:
  + Model Selection and Training.
  + Model Evaluation and Hyperparameter Tuning.
  + Prepare Model Deployment Plan.

**Deliverables:**

* Report (PDF):
  + Description of design choices.
  + Performance evaluation of the model.
  + Discussion of future work.
* Source Code:
  + Complete code used to create the pipeline.
  + Packaged in a zip file with a README for installation and execution instructions.

**Tasks/Activities List:**

* Collect the data from various resources.
* Data Preprocessing.
* Feature Engineering and Selection.
* Train/Test Split.
* Model Evaluation Metrics: Accuracy, precision, recall, F1-score.
* Model Selection, Training, Predicting, and Assessment.
* Hyperparameter Tuning/Model Improvement.
* Model Deployment Plan.

**Success Metrics:**

* Achieve an accuracy of > 85% on the test data set.
* Include methods for hyperparameter tuning.
* Perform thorough model validation.

**Bonus Points:**

* Provide a zip file with a README explaining the installation and execution of the end-to-end pipeline.
* Demonstrate the benefits of your solution through comprehensive documentation.

By following this structured approach, you will be able to develop a robust NLP-based intensity analysis system that meets the project's objectives and deliverables.

**Objective:**

Your code aims to predict emotions (not specifically intensity) in text reviews, aligning partially with the project's goal. To fully meet the objective, you would need to focus more on predicting the intensity of emotions.

**Focus Areas and Steps:**

* Data Collection:
  + Your code reads data from a CSV file. Ensure this data includes intensity labels or is suitable for deriving intensity information.
* Data Preprocessing:
  + Done: Tokenization, padding sequences, label encoding.
  + Missing: Removal of HTML tags, special characters, normalization (lowercasing, stemming, lemmatization).
* Feature Engineering:
  + Done: Tokenization and creation of padded sequences.
  + Missing: Extraction of features like n-grams, TF-IDF scores, and word embeddings.
* Model Development:
  + Done: Sequential model with embedding, dense layers.
  + Could Improve: Consider using more advanced models like LSTM or BERT for better performance.
* Testing and Validation:
  + Done: Split data into training and testing sets, basic accuracy evaluation.
  + Could Improve: Add metrics like precision, recall, and F1-score.
* Hyperparameter Tuning:
  + Missing: No hyperparameter tuning shown. Implement Grid Search or Random Search for optimization.
* Deployment:
  + Missing: No deployment code. Consider using Flask or FastAPI for real-time predictions.

**Specific Code Requirements:**

* Data Preprocessing: Ensure noise removal, normalization, and handling of missing values:

Code

def clean\_text(text):

text = re.sub(r'<.\*?>', '', text) # Remove HTML tags

text = re.sub(r'[^a-zA-Z\s]', '', text) # Remove special characters

text = text.lower() # Lowercase

text = ' '.join(word for word in text.split() if word not in eng\_stopwords) # Remove stopwords

return text

data['DATA'] = data['DATA'].apply(clean\_text)

* Feature Engineering: Consider extracting TF-IDF features or word embeddings:

Code

from sklearn.feature\_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(max\_features=5000)

X\_tfidf = tfidf.fit\_transform(data['DATA'])

* Model Selection and Training: For better results, use an advanced model like LSTM:

Code

from keras.layers import LSTM

model = Sequential()

model.add(Embedding(input\_dim=len(tokenizer.word\_index) + 1, output\_dim=128, input\_length=max\_length))

model.add(LSTM(128, return\_sequences=True))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(len(one\_hot\_labels[0]), activation='softmax'))

* Model Evaluation: Add precision, recall, and F1-score:

Code

from sklearn.metrics import classification\_report

y\_pred = model.predict(xtest)

y\_pred\_labels = np.argmax(y\_pred, axis=1)

y\_true\_labels = np.argmax(ytest, axis=1)

print(classification\_report(y\_true\_labels, y\_pred\_labels))

* Hyperparameter Tuning: Use Grid Search for hyperparameter tuning:

Code

from sklearn.model\_selection import GridSearchCV

# Example with an SVM

from sklearn.svm import SVC

param\_grid = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}

grid = GridSearchCV(SVC(), param\_grid, refit=True, verbose=2)

grid.fit(xtrain, ytrain)

print(grid.best\_params\_)

* Deployment: Implement a simple Flask API:

Code

from flask import Flask, request, jsonify

app = Flask(\_\_name\_\_)

@app.route('/predict', methods=['POST'])

def predict():

text = request.json['text']

sequence = tokenizer.texts\_to\_sequences([text])

padded\_sequence = pad\_sequences(sequence, maxlen=max\_length)

prediction = model.predict(padded\_sequence)

predicted\_label = label\_encoder.inverse\_transform([np.argmax(prediction[0])])

return jsonify({'emotion': predicted\_label[0]})

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)