**Integrative Task II**

**Andrés Arango**

**[Nombre]**

**Icesi University**

**Computation and Discrete Structures III**

**Cali, Colombia**

**November 24th 2023**

Introduction

As in all integrative tasks, the idea is to apply the concepts seen in class, in this one the concepts based on artificial intelligence and recurrent neural networks will be applied.

Aims

Main

1. Build a sentiment analysis model using supervised learning with vanilla Recurrent Neural Networks and LSTM

Secondaries

1. Create a database with sentences and the type of sentiment of itself.
2. Tokenize the sentences to find a way to build a supervised learning model.
3. Implement a DummyClassifier for the model.
4. Implement a vanilla RNN sentiment analysis model.
5. Implement a LSTM sentiment analysis model.

Theoretical framework

Machine Learning

Machine learning is a subfield of artificial intelligence that focuses on the development of algorithms and models that enable computer systems to learn from data and improve their performance over time without being explicitly programmed. The core idea behind machine learning is to enable computers to automatically identify patterns, make predictions, or optimize decisions based on experience or historical data. This is achieved through the utilization of mathematical and statistical techniques that allow algorithms to generalize from examples and adapt to new information. Machine learning finds applications in various domains, including image and speech recognition, natural language processing, recommendation systems, and autonomous vehicles, among others. The overarching goal is to empower machines to acquire knowledge and skills autonomously, enhancing their ability to solve complex tasks and contribute to intelligent decision-making processes.

Supervised Learning

Supervised learning is a type of machine learning paradigm where the algorithm is trained on a labeled dataset, meaning that the input data is paired with corresponding output labels. The goal of supervised learning is for the algorithm to learn a mapping from input data to the correct output by generalizing patterns from the labeled examples provided during training.

Pandas

Pandas is a popular open-source data manipulation and analysis library for Python. It provides data structures for efficiently storing and manipulating large datasets and tools for working with structured data seamlessly.

Pandas simplifies common data tasks, such as:

* Loading data from various file formats (CSV, Excel, SQL databases, etc.).
* Cleaning and preprocessing data by handling missing values, removing duplicates, and transforming data.
* Indexing, slicing, and sub setting data for easy extraction of relevant information.
* Performing statistical and mathematical operations on the data.
* Merging and joining datasets.
* Time-series analysis and handling.

Pandas is widely used in data science, machine learning, and other domains where data manipulation and analysis are essential. Its intuitive and powerful functionality makes it a go-to library for working with structured data in Python.

Sklearn

Scikit-learn, often abbreviated as sklearn, is a popular open-source machine learning library for Python. It provides a simple and efficient set of tools for data analysis and modeling, with a focus on classical machine learning algorithms. Scikit-learn is built on NumPy, SciPy, and Matplotlib, making it compatible with other scientific computing libraries in the Python ecosystem.

NLTK

NLTK, or Natural Language Toolkit, is a powerful library for working with human language data (text) in Python. It provides easy-to-use interfaces to perform various natural language processing (NLP) tasks. NLTK is widely used in research and education and serves as a valuable resource for developers and researchers working on projects involving textual data.

Key features and functionalities of NLTK include:

* Text Processing: NLTK offers tools for tokenization, stemming, lemmatization, and other text processing tasks. These tools allow you to break down text into meaningful components and standardize word forms.
* Part-of-Speech Tagging: NLTK includes modules for part-of-speech tagging, which involves labeling words in a text with their respective parts of speech (e.g., noun, verb, adjective).
* Named Entity Recognition: NLTK provides functions for identifying and classifying named entities (e.g., names of people, organizations, locations) in text.
* Sentiment Analysis: NLTK supports sentiment analysis, allowing you to determine the sentiment (positive, negative, neutral) expressed in a piece of text.
* Corpus and Resources: NLTK includes a variety of corpora and lexical resources for different languages. These resources are useful for training and testing NLP models.
* Machine Learning with Text Data: NLTK integrates with machine learning libraries and provides functionalities for feature extraction and classification of text data.

Dummy Classifier

Dummy Classifier is a simple baseline classifier that is often used for comparison purposes. It serves as a basic reference point for evaluating the performance of more sophisticated machine learning models. The Dummy Classifier makes predictions using simple rules and is particularly useful for assessing whether a more complex model provides significant improvements over a naive or random strategy.

RNN Vanilla

A vanilla Recurrent Neural Network (RNN) is a type of neural network architecture designed to work with sequential data, such as time series or natural language. Unlike traditional feedforward neural networks, which process input data in isolation, RNNs have connections that form a directed cycle, allowing them to maintain a memory of previous inputs in their internal state.

In a vanilla RNN, the key idea is to capture dependencies and patterns in sequential data by maintaining a hidden state that evolves as the network processes each element in the sequence. The hidden state serves as a form of memory, allowing the network to consider context from previous time steps when making predictions or generating output.

However, vanilla RNNs have limitations. They struggle with capturing long-range dependencies in sequences, a problem known as the "vanishing gradient" problem. As the network processes input over time, gradients can diminish, making it challenging to learn from earlier parts of the sequence.

LSTM

A Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture long-range dependencies in sequential data. LSTMs are particularly effective in tasks involving time series prediction, natural language processing, and other applications with sequential patterns.

The key innovation of LSTMs lies in their ability to maintain a cell state, which acts as a memory unit that can selectively remember or forget information over long sequences. This enables LSTMs to capture and retain relevant information from earlier time steps, making them well-suited for tasks that involve understanding context over extended periods.

The architecture of an LSTM cell includes three interacting gates:

1. Forget Gate: Determines what information from the cell state should be discarded or kept. It regulates the flow of information from the previous cell state.
2. Input Gate: Decides what new information should be stored in the cell state. It involves updating the cell state with new candidate values.
3. Output Gate: Controls what information from the cell state should be used to generate the output. It influences the prediction or the hidden state of the LSTM.

The gating mechanisms in LSTMs enable them to mitigate the vanishing gradient problem by selectively updating and passing information through the network. This makes LSTMs more effective at capturing dependencies in sequential data over longer distances than traditional RNNs.

In summary, LSTMs are a type of RNN designed to capture long-term dependencies in sequential data through the use of memory cells and gating mechanisms, making them well-suited for a wide range of applications involving sequences and temporal patterns.

GridSearchCV

GridSearchCV is a function in the scikit-learn library for Python that performs an exhaustive search over a specified parameter grid for a machine learning algorithm. It is used for hyperparameter tuning, where the goal is to find the best combination of hyperparameter values that maximizes the performance of a model.

How GridSearchCV works:

* Parameter Grid: You define a grid of hyperparameter values that you want to explore. Each point in this grid represents a combination of hyperparameters that you want to test.
* Cross-Validation: GridSearchCV uses cross-validation to evaluate the performance of each combination of hyperparameters. It splits the training data into multiple subsets (folds), trains the model on some folds, and evaluates it on the remaining folds. This process is repeated for each combination of hyperparameters.
* Model Training and Evaluation: For each combination of hyperparameters, the model is trained and evaluated using cross-validation. The performance metric (such as accuracy, precision, or others) is computed for each combination.
* Best Hyperparameters: After the exhaustive search, GridSearchCV identifies the combination of hyperparameters that yielded the best performance according to the specified metric.

Methodology

The methodology for this project is the same of a machine learning project, this one follows these steps:

1. Define the Problem:

* Clearly define the problem you want to solve. Understand the goals and objectives of the project.
* Define the scope and constraints of the problem.

1. Gather Data:

* Collect relevant data for your problem. This may involve acquiring datasets from various sources or generating your own data.
* Ensure that the data is representative of the problem and is of sufficient quality.

1. Explore and Prepare the Data:

* Perform exploratory data analysis (EDA) to understand the characteristics of the data.
* Handle missing values, outliers, and other data preprocessing tasks.
* Encode categorical variables, scale numerical features, and perform other necessary transformations.

1. Feature Engineering:

* Create new features that might improve the performance of your model.
* Select relevant features and discard irrelevant ones.

1. Split the Data:

* Split the dataset into training and testing sets. The training set is used to train the model, while the testing set is used to evaluate its performance.

1. Choose a Model:

* Select a machine learning algorithm that is suitable for your problem. Consider factors like the type of problem (classification, regression, etc.) and the characteristics of your data.

1. Train the Model:

* Train the chosen model using the training dataset.
* Fine-tune hyperparameters to optimize model performance. Use techniques like cross-validation.

1. Evaluate the Model:

* Assess the model's performance on the testing set using appropriate evaluation metrics (accuracy, precision, recall, F1 score, etc.).
* Analyze the results and iterate on the model or data preprocessing if necessary.

1. Hyperparameter Tuning:

* Conduct further hyperparameter tuning to improve model performance. This may involve using techniques like grid search or random search.

1. Deploy the Model:

* If the model meets the desired performance, deploy it to a production environment. This may involve creating APIs, integrating with existing systems, or deploying as a web service.

1. Document the Project:

* Document all aspects of the project, including the problem definition, data sources, preprocessing steps, model selection, training process, and evaluation results. This documentation is valuable for future reference and collaboration.