**Key Topics:**

1. **Introduction to Data Science:**
   * Definition and significance of data science in various industries.
   * The role of a data scientist and the data science lifecycle.
2. **Data Collection and Preprocessing:**
   * Methods for collecting data from various sources.
   * Data cleaning and preprocessing techniques to handle missing values, outliers, and inconsistencies.
   * Data integration and transformation processes.
3. **Exploratory Data Analysis (EDA):**
   * Techniques for summarizing and visualizing data.
   * Identifying patterns, trends, and anomalies in datasets.
   * Use of statistical methods to explore and understand data distributions.
4. **Statistical Foundations:**
   * Descriptive statistics: mean, median, mode, variance, and standard deviation.
   * Probability theory and distributions.
5. **Data Visualization:**
   * Principles of effective data visualisation.
   * Tools and libraries for creating visualisations (e.g., **Matplotlib, Seaborn**, Tableau).
   * Designing and interpreting various types of **plots, charts, and dashboards**.
6. **Introduction to Machine Learning:**
   * Basic concepts and types of machine learning: supervised, unsupervised, and reinforcement learning.
   * Key algorithms: **regression, classification, clustering, and decision trees**.


   * Model evaluation and validation techniques.
7. **Data Wrangling and Transformation:**
   * Techniques for manipulating and transforming data using libraries such as **Pandas**.
   * **Feature engineering and selection methods**to improve model performance.
8. **Big Data Technologies:**
   * Overview of big data and its significance.

   * Introduction to Hadoop, Spark, and other big data tools.
   * Processing and analysing **large-scale datasets**.
9. **Ethics and Data Privacy:**
   * Ethical considerations in data science.
   * Data privacy laws and regulations.
   * Best practices for responsible data handling and analysis.
10. **Applied Data Science Projects:**
    * Hands-on projects and case studies to apply learned concepts.
    * Working with real-world datasets to solve practical problems.
    * **Collaboration** and presentation of findings.

**Learning Outcomes:** By the end of the course, students will be able to:

* Understand and articulate the **fundamental concepts of data science**.
* Collect, preprocess, and clean data for analysis.
* **Perform exploratory data analysis** and visualise data effectively.
* **Apply statistical methods** to analyse data.
* Understand basic machine learning concepts and algorithms.
* Use data wrangling techniques to prepare data for modeling.
* Gain insights into big data technologies and their applications.
* Recognise the importance of ethics and data privacy in data science.

This is a practical class where I will have to code and explain the attached code using the UGRansome dataset.

You can download the dataset here: Nkongolo Wa Nkongolo., 2023. UGRansome dataset. Available at: https://www.kaggle.com/dsv/7172543 [Accessed 25 July 2024]. DOI: 10.34740/KAGGLE/DSV/7172543.

I recommend you to open an account on Kaggle and experiment with various datasets and code (you can use your Kaggle profile on your CV when applying for Data Science positions). Upvote the UGRansome dataset to support our work...

Kaggle is an online community and platform for data scientists and machine learning practitioners. It provides users with resources and tools to build and hone their data science and machine learning skills. Key features of Kaggle include:

Competitions: Users can participate in machine learning competitions where they solve real-world problems and compete for prizes. These competitions often provide datasets and a leaderboard to track progress.

Datasets: Kaggle hosts a vast collection of public datasets, which users can access and use for analysis, model building, and learning.

Kernels (Notebooks): Kaggle provides a cloud-based coding environment called Kernels, where users can write and run code in Python or R without needing to install any software on their local machines!

Learning: Kaggle offers courses and tutorials on various topics in data science and machine learning, allowing users to learn at their own pace.

Community: Users can engage with a global community of data scientists, participate in discussions, share code, and collaborate on projects.

Kaggle is widely used for both educational purposes and professional development, offering a platform to learn, practice, and demonstrate skills in data science and machine learning.

The code I will use for this class will demonstrate the following concepts:

1. How to read a dataset (in this case the UGRansome dataset)

2. How to name columns of the dataset

3. How to detect anomalies

4. How to remove anomalies

5. Data preprocessing

6. Data visualisation

7. Data encoding

8. Lecture 3 will continue this code ...

I have attached the code (see ugransome-machine-learning.py).

The code is also available on Kaggle.

Definition: A DataFrame is a two-dimensional, size-mutable, and potentially heterogeneous tabular data structure in the pandas library of Python. It is similar to a table in a database or an Excel spreadsheet, where data is arranged in rows and columns.

Exploratory data analysis (EDA): packages in Python is extremely beneficial for gaining insights into the dataset, cleaning and preparing the data, and guiding the feature engineering and modeling process. These packages provide a range of tools for summarizing, visualizing, and understanding the data, making them essential for any data science project.

The UGRansome dataset:

To use a module (e.g., Sweetviz), you need to install it first. You can do this using pip in the terminal.

Command to install Sweetviz: pip install sweetviz

Sweetviz is EDA module that generates a highly detailed, visual analysis of the data and provides a comprehensive visual report of the dataset (please do not incorporate vusualizations generated by Sweetviz in your assignment. Only using it to understand your data).

Code

import sweetviz as sv

report = sv.analyze(UGRansome)

report.show\_html('report.html')

These metrics—mean, median, and standard deviation—provide important insights into the characteristics of a dataset:

Mean:

Central Tendency: The mean gives you an idea of the average value in the dataset.

Overall Level: It indicates the overall level of the values in the dataset.

Effect of Outliers: Since the mean is sensitive to outliers, a few extremely high or low values can significantly affect it. This can be useful to know if the dataset contains outliers.

Median:

Central Tendency: Like the mean, the median gives you an idea of the central tendency of the dataset.

Middle Value: The median is the middle value when the data is ordered, so it represents the 50th percentile.

Resilience to Outliers: The median is not affected by outliers, making it a better measure of central tendency when the dataset contains extreme values.

Standard Deviation:

Spread: The standard deviation tells you how spread out the values in the dataset are around the mean.

Variability: A low standard deviation means the values are close to the mean, indicating low variability. A high standard deviation means the values are spread out over a wider range, indicating high variability.

Data Consistency: It helps to understand the consistency of the data. For example, in quality control processes, a high standard deviation might indicate inconsistent product quality.

What These Metrics Tell You About a Dataset

Central Tendency and Typical Value: The mean and median provide information about the central tendency or typical value of the data.

If the mean and median are close, the dataset is likely symmetrically distributed.

If the mean is significantly higher or lower than the median, the dataset may be skewed.

Skewness:

If the mean is greater than the median, the data might be right-skewed (positively skewed).

If the mean is less than the median, the data might be left-skewed (negatively skewed).

Variability and Consistency:

A low standard deviation indicates that the data points are close to the mean, suggesting low variability and high consistency.

A high standard deviation indicates that the data points are spread out over a larger range, suggesting high variability and low consistency.

Practical Examples

Income Data: In income data, the mean might be higher than the median if there are a few individuals with very high incomes (right-skewed distribution). The median might give a better sense of a "typical" income.

Test Scores: For test scores of a class, if the standard deviation is low, it indicates that most students scored similarly. A high standard deviation would indicate a wide range of scores, with students scoring very differently from each other.

Quality Control: In manufacturing, a low standard deviation of product dimensions might indicate high precision in the manufacturing process, whereas a high standard deviation might indicate issues with the process consistency.

Understanding these metrics helps in making informed decisions, identifying patterns, and detecting anomalies in the data.

A correlation matrix is a table showing the correlation coefficients between many variables. Each cell in the table shows the correlation between two variables. The value is in the range of -1 to 1.

1 indicates a strong positive correlation: as one variable increases, the other variable increases.

-1 indicates a strong negative correlation: as one variable increases, the other variable decreases.

0 indicates no correlation: the variables do not have a linear relationship.

Why Use a Correlation Matrix?

Identify Relationships: It helps in identifying the relationships between different variables in a dataset.

Feature Selection: It can be used in feature selection to identify highly correlated features and reduce redundancy.

Data Understanding: It provides a quick overview of how variables are related to each other, which is essential for data exploration and understanding.

Lecture Code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from scipy import stats

from sklearn.preprocessing import StandardScaler

# Phase 1: Data Collection (e.g., UGRansome dataset)

df2 = pd.read\_csv(r'C:/Users/u21629545/Downloads/archive (8)/final(2).csv')

df2

df2.columns = ['Time','Protocol','Flag','Family','Clusters','SeedAddress','ExpAddress','BTC','USD','Netflow\_Bytes','IPaddress','Threats','Port','Prediction']

df2

# Data cleaning

# Renaming the attack "Bonet" to "Botnet"

df2['Threats'] = df2['Threats'].str.replace('Bonet', 'Botnet')

# Print the modified DataFrame

df2

# Phase 2: Data Preparation (feature engineering and data transformation)

# --- Drop all duplicate rows --- #

df2 = df2.drop\_duplicates()

# --- Remove negative values from time/timestamp feature --- #

df2['Time'] = df2['Time'] + 11

# --- Math transformations to reduce skewness --- #

# --- Log transformation applied to column NETFLOW\_BYTES --- #

# A log transformation involves taking the natural logarithm (base e) of each data point in a particular column or feature.

#Logarithmic transformations are often used to reduce the impact of extreme values (outliers) and make the data conform more

#closely to a normal distribution. They are particularly useful when dealing with positively skewed data,

#where the tail of the distribution is elongated on the right side.

#The np.log() function is a common way to perform a logarithmic transformation in Python.

#The + 1 added to the data points is often used to avoid issues with taking the logarithm of zero or negative values.

#It's a common practice to add a small constant like 1 to the data before applying the logarithm.

#By applying a log transformation to a feature, you're essentially compressing the range of values in that feature,

#which can help in cases where the data exhibits a rightward skew, making it more suitable for certain types of analysis

#or modeling techniques that assume normally distributed data.

df2['Netflow\_Bytes'] = np.log(df2['Netflow\_Bytes']+1)

# --- Square root transformation applied to columns USD ---#

#Square Root Transformation: A square root transformation involves taking the square root of each data point in the

#specified column. In this case, it's applied to the 'USD' column.

#Square root transformations are a type of mathematical transformation used to mitigate the impact of right-skewed data.

#Just like logarithmic transformations, square root transformations can help make the data more symmetric and closer to

#a normal distribution.

#The np.sqrt() function is used to calculate the square root.

#By applying a square root transformation to the 'USD' column, the code is attempting to make the data distribution less skewed

#and more suitable for certain statistical analyses or modeling techniques that assume normally distributed data or

#require data to be more symmetric. It's a common technique used in data preprocessing to improve the quality of data for

#analysis or modeling

df2['USD'] = np.sqrt(df2['USD'])

# --- Yeo Johnson transformation applied to columns BTC--#

#Yeo-Johnson transformation is being applied to the 'BTC' column in the DataFrame (df2['BTC']).

#This transformation is used to modify the data in the 'BTC' column to make its distribution more normalized or symmetric

#The Yeo-Johnson transformation is a mathematical transformation technique used to modify the distribution of data.

#It can be applied to both positive and negative values and is more versatile than some other transformations like the Box-Cox transformation.

#The transformation is performed using the stats.yeojohnson() function from a library like SciPy

df2['BTC'], \_ = stats.yeojohnson(df2['BTC'])

#--PLOTING TRANSFORMED DATA--#

fig, ax = plt.subplots(figsize=(10, 6))

# Plot the transformed 'USD' column

ax.hist(df2['USD'], bins=50, alpha=0.5, color='blue', label='USD (Square Root)')

# Plot the transformed 'BTC' column

ax.hist(df2['BTC'], bins=50, alpha=0.5, color='green', label='BTC (Yeo-Johnson)')

# Plot the transformed 'Netflow\_Bytes' column

ax.hist(df2['Netflow\_Bytes'], bins=50, alpha=0.5, color='red', label='Netflow\_Bytes (Log)')

# Add labels and a legend

ax.set\_xlabel('Transformed Values')

ax.set\_ylabel('Frequency')

ax.set\_title('Distribution of Transformed Columns')

ax.legend()

# Show the plot

plt.show()

# Create a figure and axis for the plot

fig, ax = plt.subplots(figsize=(10, 6))

# Create a StandardScaler instance

# The StandardScaler is a common preprocessing technique used in machine learning and data analysis.

#It is used to standardize or normalize the features of a dataset by scaling them such that they have a mean of 0 and a standard

#deviation of 1.

#Standardizing the features is useful because it makes different features more directly comparable, especially in algorithms

#that are sensitive to the scale of the input data, such as many machine learning algorithms.

#In the code provided, scaler is created as an instance of the StandardScaler class, which can then be used to standardize

#the specified columns in the df2 DataFrame using the fit\_transform method, as seen in the subsequent code

scaler = StandardScaler()

# Normalize each column's features

df2\_normalized = df2.copy()

df2\_normalized[['USD', 'BTC', 'Netflow\_Bytes']] = scaler.fit\_transform(df2[['USD', 'BTC', 'Netflow\_Bytes']])

# Plot the density of the normalized 'USD' column

sns.kdeplot(df2\_normalized['USD'], color='blue', label='USD (Square Root)', ax=ax)

# Plot the density of the normalized 'BTC' column

sns.kdeplot(df2\_normalized['BTC'], color='green', label='BTC (Yeo-Johnson)', ax=ax)

# Plot the density of the normalized 'Netflow\_Bytes' column

sns.kdeplot(df2\_normalized['Netflow\_Bytes'], color='red', label='Netflow\_Bytes (Log)', ax=ax)

# Add labels and a legend

ax.set\_xlabel('Normalized Values')

ax.set\_ylabel('Density')

ax.set\_title('Density Plot of Normalized Columns')

ax.legend()

# Show the plot

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# df2 is your dataframe

plt.figure(figsize=(17, 6))

corr = df2.corr(method='spearman')

my\_m = np.triu(corr)

sns.heatmap(corr, mask=my\_m, annot=True, cmap="Set2")

plt.show()

correlation\_matrix = df2.corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.show()

#The preprocessing module in scikit-learn provides various tools and techniques for preprocessing your data before

#feeding it into machine learning models.

#This preprocessing is crucial to improve the quality of your data and the performance of your models.

from sklearn import preprocessing #pip install scikit-learn

#The code segment uses scikit-learn's LabelEncoder to transform categorical variables into numerical values.

#Each categorical column, such as 'Protocol,' 'Flag,' 'Family,' 'SeedAddress,' 'ExpAddress,' 'IPaddress,' 'Threats,' and

#'Prediction,' is encoded into unique numeric labels.

#This preprocessing step is essential for machine learning algorithms, as they typically require numerical input data

#instead of categorical labels.

lab\_encoder = preprocessing.LabelEncoder() # transformation of categorical to numeric

df2['Protocol'] = lab\_encoder.fit\_transform(df2['Protocol'])

df2['Flag'] = lab\_encoder.fit\_transform(df2['Flag'])

df2['Family'] = lab\_encoder.fit\_transform(df2['Family'])

df2['SeedAddress'] = lab\_encoder.fit\_transform(df2['SeedAddress'])

df2['ExpAddress'] = lab\_encoder.fit\_transform(df2['ExpAddress'])

df2['IPaddress'] = lab\_encoder.fit\_transform(df2['IPaddress'])

df2['Threats'] = lab\_encoder.fit\_transform(df2['Threats'])

df2['Prediction'] = lab\_encoder.fit\_transform(df2['Prediction'])

df2

Lecture 3 (Supervised and Ensemble Learning)Lecture 3 (Supervised and Ensemble Learning)

Attached Files:

File Supervised\_Learning\_Ensemble\_Learning\_RF\_NB\_SVM.pdf Supervised\_Learning\_Ensemble\_Learning\_RF\_NB\_SVM.pdf - Alternative Formats (1.62 MB)

File ugransome-machine-learning-python-notebook (1).ipynb (1.117 MB)

In this class I will use Lecture's 2 code to demonstrate the following concepts:

1. Classification and prediction using the Naive Bayes algorithm.

2. Classification and prediction using the Support Vector Machine (SVM) algorithm.

3. Classification and prediction using the Random Forest algorithm.

4. Classification and prediction using Ensemble Learning algorithm.

This is a continuation of Lecture 2's code using the UGRansome dataset.

Naive Bayes is a probabilistic classifier based on Bayes' theorem with an assumption of independence between features. It is often used for classification tasks and works particularly well for large datasets.

Bayes' Theorem: Naive Bayes uses Bayes' theorem to calculate the probability of a class given the features

Naive Assumption: The "naive" part refers to the assumption that all features are independent given the class. This simplifies the computation of P not stretchy left parenthesis X ∣ C not stretchy right parenthesisP(X∣C) to:

where xi​ are the individual features.

Types: There are different types of Naive Bayes classifiers based on the type of features:

Gaussian Naive Bayes: Assumes that features follow a Gaussian distribution.

Multinomial Naive Bayes: Used for discrete features like word counts in text classification.

Bernoulli Naive Bayes: Used for binary/boolean features.

Strengths:

Simple and easy to implement.

Works well with high-dimensional data.

Weaknesses:

The assumption of feature independence is often unrealistic, which can affect performance.

Support Vector Machine (SVM)

Support Vector Machine (SVM) is a supervised learning algorithm used for classification and regression tasks. It finds the hyperplane that best separates different classes in the feature space.

Hyperplane: SVM tries to find the optimal hyperplane that maximizes the margin between two classes. The margin is the distance between the hyperplane and the nearest data points from each class, which are called support vectors.

Kernel Trick: SVM can be extended to handle non-linear relationships using the kernel trick, which transforms the data into a higher-dimensional space where a linear hyperplane can be used to separate the classes.

Regularization: SVM includes a regularization parameter that controls the trade-off between maximizing the margin and minimizing classification error.

Strengths:

Effective in high-dimensional spaces.

Can handle non-linear boundaries using kernels.

Weaknesses:

Can be computationally expensive for large datasets.

Requires careful tuning of hyperparameters.

Random Forest

Random Forest is an ensemble learning method that combines multiple decision trees to improve classification and regression performance. It uses the following principles:

Bootstrap Aggregating (Bagging): Random forests build multiple decision trees using different subsets of the training data. Each tree is trained on a random sample of the data with replacement (bootstrap sample).

Random Feature Selection: When splitting nodes in a tree, random forests consider a random subset of features. This reduces correlation between trees and improves generalization.

Voting/Averaging: For classification, the final prediction is made by majority voting from all decision trees. For regression, it is the average of predictions from all trees.

Strengths:

Handles both classification and regression tasks.

Reduces overfitting compared to individual decision trees.

Can handle large datasets with many features.

Weaknesses:

Can be computationally intensive and slower to predict than individual decision trees.

Less interpretable than a single decision tree.

Ensemble Learning

Ensemble Learning is a machine learning paradigm that combines multiple models to produce a better overall performance than individual models. The idea is to leverage the strengths of different models and mitigate their weaknesses.

Strengths:

Often improves accuracy and robustness compared to individual models.

Can handle a wide variety of data and problems.

Weaknesses:

Can be complex to implement and tune.

May require more computational resources.

In summary:

Naive Bayes is a probabilistic classifier based on independence assumptions.

SVM is a powerful classifier that finds optimal decision boundaries.

Random Forest is an ensemble of decision trees that improves performance through averaging and randomization.

Ensemble Learning combines multiple models to improve performance and generalization.

Types of Ensemble Methods:

Bagging (Bootstrap Aggregating): Builds multiple models (e.g., decision trees) using different subsets of the data. The final prediction is an aggregation of predictions from all models. Random Forest is a popular bagging method.

Boosting: Sequentially builds models, where each new model corrects errors made by the previous ones. Examples include AdaBoost and Gradient Boosting Machines (GBM).

Stacking: Combines multiple models (base learners) and uses another model (meta-learner) to aggregate their predictions. The base learners might use different algorithms, and the meta-learner combines their outputs.

Lecture 4 (Unsupervised Learning and Deep Learning)Lecture 4 (Unsupervised Learning and Deep Learning)

Attached Files:

File 2024\_INF 491\_791\_L 04 Applied Statistical Analysis.pdf 2024\_INF 491\_791\_L 04 Applied Statistical Analysis.pdf - Alternative Formats (2.764 MB)

File unsupervisedeeplearning.ipynb (267.754 KB)

File KMEANS-IRIS-DATA.pdf KMEANS-IRIS-DATA.pdf - Alternative Formats (564.046 KB)

File Kmeans\_code.ipynb (340.862 KB)

In this class I will demonstrate the following concepts:

1. Unsupervised Learning for Image Processing.

2. Deep Neural Networks Architectures (RNN, CNN, and LSTM).

Dataset: MNIST (Image Dataset).

Recaputilation: Lazypredictor and LazyRegressor model for supervised learning and ROC evaluation metric.

See this link, it teachs you how to plot the ROC value using the UGRansome dataset https://medium.com/@samson.sabu/blog-post-ransomware-analysis-using-machine-learning-and-deep-learning-a-comprehensive-study-with-a48dfe024dbf

Extra marks is given to students who can execute Lazy predict and Lazy regressor models on the gameplay dataset and compare the results with the ensemble learning.

Extra marks is given to students who will evaluate the gameplay dataset with the ROC values.

I will not share the lazy predict and regressor code.

The ROC code is given.

ROC stands for Receiver Operating Characteristic. It's a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.:

True Positive Rate (TPR) or Sensitivity: This is plotted on the y-axis. It represents the proportion of actual positives correctly identified by the model (e.g., the percentage of cryptojacking cases correctly detected).

False Positive Rate (FPR): This is plotted on the x-axis. It represents the proportion of actual negatives incorrectly identified as positives (e.g., the percentage of normal cases incorrectly flagged as cryptojacking).

The ROC curve shows the trade-off between sensitivity and specificity (1 - FPR). A model with a curve closer to the top-left corner indicates a better performance, as it shows a higher true positive rate with a lower false positive rate.

The area under the ROC curve (AUC) is often used as a single metric to summarize the model's performance. An AUC of 1.0 represents a perfect model, while an AUC of 0.5 suggests no discrimination, akin to random guessing.

Unsupervised learning for image processing involves analyzing and extracting patterns from images without using labeled data. Unlike supervised learning, where models are trained on labeled datasets (i.e., images with corresponding labels), unsupervised learning works with unlabeled data, discovering hidden structures and relationships in the data. Here are some common techniques and their applications in image processing:

Clustering

Technique: Clustering algorithms like K-means, hierarchical clustering, and DBSCAN group similar images or regions within images into clusters based on pixel intensity, color, texture, or other features.

Applications:

Image segmentation (dividing an image into meaningful segments)

Organizing large image datasets by grouping similar images together

Anomaly detection in images by identifying clusters of unusual patterns

Dimensionality reduction

Technique: Techniques like Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), and autoencoders reduce the dimensionality of image data while preserving important information.

Applications:

Image compression by reducing the size of image data

Visualization of high-dimensional image data in 2D or 3D space

Feature extraction for downstream tasks like classification or clustering

Autoencoders

Technique: An autoencoder is a type of neural network used to learn efficient representations of data, often for the purpose of dimensionality reduction or feature learning. It consists of an encoder that compresses the image into a latent space and a decoder that reconstructs the image from this space.

Applications:

Image denoising by training autoencoders to reconstruct clean images from noisy ones

Anomaly detection by comparing the input image with its reconstruction

Generating new images by sampling from the latent space

Generative models

Technique: Models like Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) generate new images that resemble the original data by learning the underlying distribution of the image data.

Applications:

Image synthesis to create new, realistic images

Data augmentation to generate additional training data for supervised learning tasks

Style transfer by altering the appearance of images to match a desired style

Self-organizing maps (SOMs)

Technique: SOMs are a type of neural network that map high-dimensional image data onto a lower-dimensional grid while preserving the topological structure of the data.

Applications:

Visualizing patterns in image datasets

Clustering similar images

Dimensionality reduction for feature extraction

Benefits

No Label Dependency: Unsupervised learning doesn't rely on labeled data, making it useful in scenarios where labeling is expensive or impractical.

Exploratory Analysis: It helps in exploring and understanding the underlying structure of image data, leading to insights that may not be apparent with supervised learning.

Feature Learning: It can discover useful features or representations of images that can be used in other tasks like classification, object detection, or segmentation.

Unsupervised learning for image processing is a powerful approach for tasks where labeled data is scarce, expensive, or unavailable. It is widely used in exploratory data analysis, feature extraction, and anomaly detection in various image processing applications.

We will implement unsupervised deep learning techniques, specifically using autoencoders with both Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), including Long Short-Term Memory (LSTM) networks.

Breakdown of the components in our architecture

Load the MNIST dataset. The MNIST dataset is a classic dataset used for training various image processing systems. It consists of handwritten digits from 0 to 9.

Normalize the data. Normalization scales the data to a standard range, usually [0, 1], which helps improve the performance and convergence of neural networks.

Reshape data for CNN (28x28x1) and RNN (28 timesteps, 28 features):

For CNN, the MNIST images are reshaped to have a single channel (grayscale) with dimensions 28x28.

For RNN, the data is reshaped to 28 timesteps, each with 28 features. This transformation considers each row of the image as a timestemp and each pixel in that row as a feature.

Build CNN and RNN based autoencoder models with encoder & decoder:

CNN-based autoencoder. Uses convolutional layers in the encoder and decoder parts to learn spatial features from the images.

RNN-based autoencoder. Uses RNN layers (or LSTM layers) to capture temporal dependencies in the image data. In this case, LSTM autoencoders are also mentioned, which use LSTM cells for encoding and decoding.

Reconstruct images using both models. After training, the autoencoders will reconstruct the input images. The CNN autoencoder will output reconstructed images based on learned spatial features, and the RNN autoencoder will output images based on learned temporal features.

Plot original and reconstructed images:

Display original: Shows the original MNIST images.

Display CNN reconstructed: Shows the images reconstructed by the CNN autoencoder.

Display RNN reconstructed: Shows the images reconstructed by the RNN autoencoder.

Show(): This command displays the plots.

This architecture involves training autoencoders with both CNN and RNN (LSTM) components, which is a common deep learning approach for image reconstruction and anomaly detection tasks.

Deep neural networks (DNNs) are a class of machine learning models composed of multiple layers of interconnected nodes (neurons) that learn to transform input data into output predictions. Different architectures of DNNs are designed to handle various types of data and tasks. Here are three common types of DNN architectures: Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Long Short-Term Memory networks (LSTMs).

Convolutional neural networks (CNNs)

Purpose: CNNs are specialized for processing grid-like data, such as images.

Key Components:

Convolutional Layers: Apply convolutional filters to detect features like edges, textures, and patterns in the input image. These filters slide over the image, producing feature maps.

Pooling Layers: Reduce the spatial dimensions of the feature maps, retaining important information while reducing computational complexity. Max pooling is a common pooling technique.

Fully Connected Layers: Neurons in these layers are fully connected to all activations in the previous layer, and they often serve as the final layers in a CNN, where classification or regression is performed.

Applications:

Image classification (e.g., identifying objects in images)

Object detection (e.g., detecting and localizing objects within an image)

Image segmentation (e.g., dividing an image into regions with different objects)

Video analysis and image generation (e.g., GANs)

The Convolutional layer applies filters to the input image to extract features, the Pooling layer downsamples the image to reduce computation, and the fully connected/dense layer makes the final prediction. The network learns the optimal filters through backpropagation and gradient descent.

Recurrent neural networks (RNNs)

Purpose: RNNs are designed for sequential data, where the order of inputs matters, such as time series, text, or speech.

Key Components:

Recurrent Layers: In an RNN, the output from the previous time step is fed back into the network along with the new input. This allows the network to maintain a memory of previous inputs, making it suitable for tasks involving sequences.

Hidden States: The hidden state in an RNN captures information about previous inputs in the sequence. It gets updated at each time step, influencing the output.

Applications:

Natural language processing (NLP) tasks like text generation, sentiment analysis, and machine translation

Time series forecasting (e.g., predicting stock prices or weather patterns)

Speech recognition and generation

Sequential data classification (e.g., activity recognition in sensor data)

Long short-term memory networks (LSTMs)

Purpose: LSTMs are a specialized type of RNN designed to better capture long-term dependencies in sequential data.

Key Components:

Memory Cell: The core of an LSTM is the memory cell, which retains information over time. It is regulated by three gates:

Input Gate: Controls how much of the new input to store in the memory cell.

Forget Gate: Decides what information to discard from the memory cell.

Output Gate: Determines the output of the LSTM at the current time step based on the memory cell's state.

Gating Mechanism: The gates use sigmoid functions to control the flow of information, ensuring that relevant information is retained and irrelevant information is discarded.

Applications:

Long-term sequence prediction (e.g., language modeling, text generation)

Machine translation (e.g., translating sentences from one language to another)

Speech synthesis and recognition

Video analysis and anomaly detection in sequential data

Key Differences and Applications

CNNs: Best suited for spatial data like images and videos, focusing on local feature detection through convolution.

RNNs: Ideal for sequential data where the order of inputs is crucial. RNNs are suitable for tasks where previous context influences the current output, such as in text or time series.

LSTMs: A type of RNN that excels at capturing long-term dependencies in sequences, making it particularly effective for tasks like language modeling and speech recognition.

Each of these architectures has unique strengths and is applied in different contexts, depending on the nature of the data and the task at hand.

MNIST Dataset for Handwriting Recognition

To reconstruct images using both CNN and RNN models, we'll modify the architecture of each model to include a decoder component. This allows the model to reconstruct the input images from the encoded representation.

Code for image reconstruction using CNN and RNN

1. Load the MNIST dataset

2. Normalize the data

3. Reshape data for CNN (28x28x1) and RNN (28 timesteps, 28 features)

4. Build:

CNN, RNN based autoencoder model with encoder & decoder

5. Reconstruct images using both models

6. Plot original and reconstructed images

6.1. Display original

6.2. Display CNN reconstructed

6.3. Display original

7. Display RNN reconstructed

8. Show()

Explanation

CNN Autoencoder:

The CNN model first encodes the input image into a lower-dimensional representation using a convolutional layer followed by max pooling.

It then reconstructs the image using upsampling and convolutional layers.

RNN Autoencoder:

The RNN model treats each row of the image as a sequence and encodes it into a fixed-size vector.

It then reconstructs the image by repeating the encoded vector and using another RNN layer to decode it back into the original shape.

Training:

Both models are trained to minimize the reconstruction loss, which measures the difference between the original and reconstructed images.

Visualization:

After training, we visualize the original and reconstructed images for both CNN and RNN models.

How to run

Install Required Libraries: Ensure you have TensorFlow installed in your environment

pip install tensorflow

Review the results. The script will display the original images along with their reconstructions from both the CNN and RNN models. You can compare how well each model reconstructs the images.

This will give you a clear comparison of how CNNs and RNNs handle image reconstruction.

cnn\_autoencoder.summary(): Prints a summary of the CNN autoencoder model, including layer types, output shapes, and the number of parameters.

rnn\_autoencoder.summary(): Prints a summary of the RNN autoencoder model, showing similar details.

lstm\_autoencoder.summary(): Prints a summary of the LSTM autoencoder model.

This will give you a detailed look at the number of layers, layer types, and other important characteristics of each model.

To include LSTM in the image reconstruction task, we'll modify the RNN model to use LSTM layers instead. LSTMs are better suited than simple RNNs for capturing long-term dependencies, which might lead to better reconstruction performance.

Code for Image Reconstruction Using CNN, RNN, and LSTM

To include LSTM in the image reconstruction task, we'll modify the RNN model to use LSTM layers instead. LSTMs are better suited than simple RNNs for capturing long-term dependencies, which might lead to better reconstruction performance.

1. Load the MNIST dataset

2. Normalize the data

3. Reshape data for CNN (28x28x1) and RNN (28 timesteps, 28 features)

4. Build:

CNN, RNN based autoencoder model with encoder & decoder

4.1. Build LSTM autoencoder with encoder & decoder

5. Reconstruct images using both models

6. Plot original and reconstructed images

6.1. Display original

6.2. Display CNN reconstructed

6.3. Display original

7. Display RNN reconstructed

8. Show()

Note: Use the uploaded code to practice.

See the K-Means code for unsupervised learning

Lecture 5 (Natural Language Processing and Computational Lexicography)Lecture 5 (Natural Language Processing and Computational Lexicography)

Attached Files:

File Last\_Year\_Lecture slide.pdf Last\_Year\_Lecture slide.pdf - Alternative Formats (1.96 MB)

File VADERCODE.ipynb (3.214 KB)

File This\_Year\_NLP.pdf (2.47 MB)

The study of natural language processing (NLP) encompasses a wide array of essential concepts and tools aimed at understanding and processing human language. Key components include lexicons, which are vital resources containing words or phrases categorized with their sentiment or other linguistic attributes. These lexicons serve as foundational tools for sentiment analysis, a crucial task in NLP that involves assessing the emotional tone conveyed by text. Techniques like bag of words and tokenization are fundamental preprocessing steps that break down text into manageable units, allowing for analysis and modeling. Moreover, part of speech tags and named entity recognition enable the identification and categorization of words based on their grammatical roles or as specific entities like names or locations within text data.

In practical applications, NLP frameworks such as the Natural Language Toolkit (NLTK) and indexed techniques like BM25 play pivotal roles. NLTK provides a comprehensive suite of libraries and algorithms for NLP tasks, facilitating tasks from tokenization to syntactic analysis. Meanwhile, BM25 indexing enhances search efficiency by ranking documents based on relevance to a query within preprocessed patterns. Addressing challenges like polysemic dilemmas, where words have multiple meanings depending on context, and managing stop words, common but non-informative words, are critical to refining NLP models and improving accuracy in tasks like text translation and sentiment analysis. Sentiment analysis itself relies on specialized lexicons like Sentistrength, VADER, SentiWordNet, Liu and Hu lexicon, AFINN-111 , and SentiWordNet, each tailored to capture nuances in sentiment expressed through text, supporting more nuanced and accurate analysis of textual data.

Computational lexicography is the field that applies computational techniques to the creation, analysis, and management of dictionaries and lexical resources. It combines principles from linguistics, computer science, and lexicography to automate the process of collecting, organizing, and defining words and their meanings. This includes tasks such as word sense disambiguation, identifying part-of-speech tags, analyzing word usage patterns, and developing algorithms that can process large corpora of text to extract and update dictionary entries efficiently.

In computational lexicography, tools like machine learning, natural language processing (NLP), and databases are employed to construct and maintain digital dictionaries, often in multiple languages. These resources are essential for various applications, such as language learning, machine translation, and text analysis.

The integration of automated methods such as Large Language Models (LLM) and GenAI allow for the rapid updating of dictionaries, the inclusion of new words, and the identification of word relationships in vast datasets, ultimately improving the accuracy and usability of lexical resources in the digital age.

Please, read the following research papers, to familiarize yourself with the NLP jargon:

Nkongolo Wa Nkongolo, M., 2023. News Classification and Categorization with Smart Function Sentiment Analysis. International Journal of Intelligent Systems, 2023(1), p.1784394. https://doi.org/10.1155/2023/1784394

Mohammed, I. and Prasad, R., 2024. Lexicon dataset for the Hausa language. Data in Brief, 53, p.110124. https://doi.org/10.1016/j.dib.2024.110124

Lecture 6 (XAI, and LLM)Lecture 6 (XAI, and LLM)

Attached Files:

File Accident \_Dataset (1).xlsx (1.75 MB)

File XAI\_LLMs.pdf XAI\_LLMs.pdf - Alternative Formats (696.808 KB)

File XAI\_LLM\_DATASCIENCE.pdf XAI\_LLM\_DATASCIENCE.pdf - Alternative Formats (838.152 KB)

XAI

LLM

📔 2023 Kaggle AI Report - Generative AI ⚡️

Meta | Llama 3.2 | Kaggle

Assignment 4 (DUE DATE 1 NOVEMBER 2024): Explainable AI (XAI) on Road Traffic Infringement Data

Objective

In this assignment, you will apply Explainable Artificial Intelligence (XAI) techniques to a dataset containing road traffic infringement data. The goal is to build and analyse a machine learning model that predicts traffic violations and provides explainable insights into the factors influencing these predictions.

Note: To improve the quality of this assignment, you are allow to work in a group of 3 persons.

Dataset

You will are provided with a dataset containing records of road traffic violations (see the attached dataset). This dataset includes various attributes, including:

Infringement ID: Unique identifier for each violation.

Vehicle Type: Type of vehicle involved (e.g., car, truck, motorcycle).

Violation Type: Type of traffic violation (e.g., speeding, parking, signal jump).

Driver Age: Age of the driver.

Location: The location where the violation occurred.

Time of Violation: The time at which the violation was recorded.

Weather Condition: Weather at the time of the violation.

Fine Amount: The monetary penalty imposed for the violation.

Tasks

Data preprocessing

Handle missing data, and inconsistency (if any).

Encode categorical variables (e.g., Vehicle Type, Violation Type).

Scale numerical features (e.g., Driver Age, Fine Amount).

Model building

Train two machine learning model (Decision Tree and Logistic Regression) to predict traffic violations.

Split the dataset into training and testing sets for model validation.

Evaluate model performance using metrics such as accuracy, precision, recall, ROC, computational time, Kappa, and F1-score.

XAI techniques

Apply XAI techniques like LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations) to explain the model's predictions. Compare the two XAI models.

For a few selected samples, provide detailed explanations of the model’s decision-making process, including the contribution of each feature to the prediction.

Visualisation

Create visualisations that represent the feature importance and the factors influencing predictions.

Use visualisation tools like confusion matrices, feature importance plots, and LIME Vs. SHAP plots to explain model behavior.

Report

Write a report summarising your findings, including the following sections:

Introduction. Briefly describe the dataset and problem statement.

Methodology. Explain the data preprocessing, model selection, and the XAI techniques applied.

Results. Present your model’s performance metrics and provide an analysis of the explainability results.

Conclusion. Discuss the insights gained from using XAI and how it helps to understand the model’s decisions regarding traffic violations.

Deliverables

The Python code (or Jupyter Notebook) used for preprocessing, model building, and explainability analysis.

A report (PDF) summarising the work done, the model performance, and the explainability analysis.

Grading Criteria

Data Preprocessing: 20%

Model Performance: 30%

XAI Analysis: 30%

Visualisation & Report: 20%

Recommended reading:

Adeliyi, T.T., Oluwadele, D., Igwe, K., Aroba, O.J. (2023). Analysis of road traffic accidents severity using a pruned tree-based model. International Journal of Transport Development and Integration, Vol. 7, No. 2, pp. 131-138. https://doi.org/10.18280/ijtdi.070208

Good luck!

Dr. WA NKONGOLO MIKE NKONGOLO

Last Lecture 7 (Ethics, Conclusion, Exam Scope)Last Lecture 7 (Ethics, Conclusion, Exam Scope)

Lecture content: Ethics in data science

1. Introduction to ethics in data science

Definition of Ethics: Overview of ethical principles and their importance in decision-making.

Privacy: Protecting individuals' personal data from unauthorized access or misuse.

Bias and Fairness: Ensuring data and algorithms do not disproportionately disadvantage any group.

Transparency: Being clear about how data is collected, processed, and how decisions are made by algorithms.

Accountability: Taking responsibility for the impact of data-driven decisions and ensuring that harm is minimized.

Security: Safeguarding data from breaches and ensuring the confidentiality and integrity of information.

Ethics in Data Science refers to the application of moral principles and values to the processes and practices involved in collecting, analyzing, and using data. It ensures that data science activities are conducted in a way that respects individual rights, promotes fairness, maintains transparency, and avoids harm. Key ethical considerations include:

Ethics in data science aims to balance innovation and societal good with legal and moral responsibilities, preventing misuse of data and ensuring trust in data-driven technologies.

Relevance to Data Science: Discuss why ethics is critical in data-driven practices, particularly due to the influence of data science on society (e.g., AI, automation).

Examples of Ethical Issues: Misuse of personal data, biased algorithms, and lack of transparency.

2. Key ethical principles in data science

Fairness: Ensuring that data models and algorithms treat all individuals or groups fairly without bias (e.g., gender, race, socio-economic status).

Transparency: Data scientists must provide transparency in their methods, data sources, and algorithms.

Accountability: Who is responsible when a data model causes harm? The need for accountability mechanisms for decisions made by AI or ML models.

Privacy: Understanding privacy laws and ensuring that personal data is collected, processed, and stored with proper consent and security measures.

Security: Ethical responsibilities in protecting data from breaches and ensuring the confidentiality, integrity, and availability of data.

3. Bias in data collection and modeling

Types of Bias: Selection bias, confirmation bias, algorithmic bias, and measurement bias.

Impact of Bias: The real-world consequences of biased algorithms, such as unfair hiring practices, biased sentencing in the judicial system, or unequal loan approval rates.

Mitigating Bias: Approaches to identify and mitigate bias, such as diverse data collection, algorithm auditing, and fairness metrics.

4. Privacy and consent in data usage

Understanding Personal Data: Define personal, sensitive, and anonymized data. Discuss the ethical considerations for handling each type.

Informed Consent: The importance of obtaining clear, informed consent from users before collecting or using their data.

Data Ownership: Who owns the data? Rights of individuals versus organizations.

Privacy Laws and Regulations: Overview of global privacy laws such as GDPR (General Data Protection Regulation) in Europe, CCPA (California Consumer Privacy Act), HIPAA (Health Insurance Portability and Accountability Act), and their implications for data science projects.

5. Ethical considerations in AI and machine learning

Explainability and Interpretability: Ethical importance of building models that are interpretable and understandable to non-experts.

Autonomous Decision-Making: Challenges with autonomous systems making decisions without human oversight (e.g., self-driving cars, automated hiring).

Ethical AI Frameworks: Overview of existing frameworks, such as AI ethics principles from Google, Microsoft, and other organizations.

Bias in Machine Learning Models: How biased data can lead to biased predictions, and ways to address these challenges.

6. Case studies in data science ethics (These case studies will be use in the Exam theory section)

Case Study 1: Cambridge Analytica and Facebook Data Scandal: Discuss how user data was exploited for political purposes without informed consent, leading to ethical violations.

Case Study 2: Amazon’s Biased Hiring Algorithm: Explore the ethical implications of building hiring algorithms that discriminated against women.

Case Study 3: Healthcare Algorithms: Ethical considerations in healthcare AI, such as algorithms misdiagnosing or prioritizing treatment unfairly based on biased data.

Case Study 4: Predictive Policing: The ethical concerns of using AI in law enforcement and the risk of amplifying systemic bias.

7. Ethical data science governance and guidelines

Data Governance Frameworks: Key components of effective data governance, including roles, policies, and standards for ethical data use.

Corporate Social Responsibility (CSR): How organizations can integrate ethical practices into their data science strategies?

8. Ethical challenges in big data and data-driven decision making

The Role of Big Data: Ethical considerations when working with massive datasets, such as the risk of re-identifying individuals in anonymized datasets.

Decision-Making in Automated Systems: Ethical implications of allowing automated systems to make decisions without human intervention, especially in critical domains like healthcare or finance.

Surveillance and Data Exploitation: The balance between data collection for societal benefits (e.g., public health) and the risk of mass surveillance.

9. Ethics in research and publication

Research Integrity: Ethical standards for publishing data science research, including transparency in methodology, reproducibility, and avoidance of cherry-picking results.

Bias in Peer Review and Publication: Ethical concerns about bias in research publication, the influence of funding sources, and the importance of diversity in research.

10. Tools and techniques for ethical data science

Bias Detection Tools: Introducing tools and techniques to detect and mitigate bias in datasets and algorithms (e.g., IBM AI Fairness 360).

Privacy-Preserving Techniques: Discuss techniques like differential privacy, federated learning, and encryption methods that allow analysis without compromising individual privacy.

Ethical AI Toolkits: Overview of toolkits designed to support ethical AI development (e.g., Google’s What-If Tool).

11. Conclusion and future trends

Emerging Trends: Discuss new challenges on the horizon (e.g., ethics in deepfakes, AI in military applications, and brain-computer interfaces).

Responsibilities of Data Scientists: Reinforce the ethical responsibility that data scientists bear in shaping society through the tools and models they develop.

The Path Forward: Think critically about how you can contribute to ethical practices in your own work.