Intrusion Detection Naiive Bayes Estimation with experimenting PCA (Milestone 3)

Amr Mohsen, 58-21006

Abdulrahman Waleed, 58-2089

Omar Khaled, 58-6794

Supervisors:

Prof. Dr. Mohamed Ashour

Eng. Youssef Tawfilis

Contents

1 Introduction **2**

1.1 An Overview 2

1.2 Task 1: From Scratch Estimation Across Test Dataset 2

1.3 Task 2: Categorical Feature Encoding and Machine Learning Model Evaluation 2

1.4 PCA techniques to compress and simplify the data 2

2 Overview for Tasks 1 and 2 **3**

3 General code **3**

3.1 Importing Libraries 3

3.2 Data Preprocessing 3

3.3 Documenting Best-Fit Distributions 4

3.4 Conditioned Data Preparation 4

4 Task 1 code **5**

4.1 Calculating PDF/PMF 5

4.2 Naïve Bayes Anomaly Detection 5

4.3 Results 6

5 Task 2 code **7**

5.1 Encoding Categorical Features 7

5.2 Model Training and Evaluation 7

5.3 Results 9

6 Overview PCA MS 3 EXTRA **10**

6.1 **Key Concepts:** 10

6.2 **Steps in PCA:** 10

6.3 **Usage in this milestone** 10

7 general PCA code **11**

7.1 Imports 11

7.2 Data preparation 11

8 Key Differences in PCA version **11**

8.1 Speed and computing 11

8.2 Predictions 12

9 PCA Results **12**

9.1 Task 1 results 12

9.2 Task 2 results 12

10 Challenges **13**

11 Conclusion **14**

12 References **14**

1 Introduction

1.1 An Overview

This milestone was built on 3 different parts (Task 1, Task 2, PCA).
The first and the second tasks are required but the thirs is extra and it was added to repeat tasks 1 and 2 but with improvements in the performance metrics.

1.2 Task 1: From Scratch Estimation Across Test Dataset

In this task, the goal is to predict anomalies in a dataset based on various features. Using Naïve Bayes, the probability of an anomaly for each row is calculated using conditional probabilities of numerical and categorical features. The process involves:

- Calculating conditional probabilities for each feature given the anomaly label.
- Assuming independence between features, combining these probabilities to predict whether an anomaly occurs or not for each row.
- Comparing the model's predictions with actual labels to evaluate its accuracy, precision, and recall.

1.3 Task 2: Categorical Feature Encoding and Machine Learning Model Evaluation

This task focuses on handling (preparing) categorical features and evaluating different Naïve Bayes models:

- **One-hot encoding** is used to transform categorical features into numerical values.
- Various Naïve Bayes models, such as **GaussianNB**, **MultinomialNB**, and **BernoulliNB**, are trained using Scikit-Learn.
- The performance of each model is assessed using **accuracy**, **precision**, and **recall** to determine which model provides the best results.
- The best-performing model is selected and justified based on these metrics.

The data of the main task 1 and 2 is stored under a file called **Mielstone 3.py**.

1.4 PCA techniques to compress and simplify the data

This extra part works on compressing the data by getting new columns from existing columns. These actions resulted in more simple calculations and faster fitting of the data.

- The columns of the main data frame are compressed to a smaller number.
- Task 1 and 2 were repeated using the data after the PCA application.
- The performance of each model is assessed using **accuracy**, **precision**, and **recall** to test if it really improves the metrics or not.
- The results of this part are under a different file called **PCA MS 3 EXTRA.py**.

2 Overview for Tasks 1 and 2

- The data in both tasks are the normal train and test data consisting of 41 columns each including the class column.

- The tasks was performed on the train first to get the best-fit distributions to be used later on the test results to evaluate if the training results were good enough or not.

- At the end, the performance metrics were evaluated using the class column and comparing it to the predictions we came out with depending on the training data.

3 General code

3.1 Importing Libraries

The script starts by importing necessary libraries and functions from past files of different milestones:

```python
import numpy as np
import pandas as pd
import scipy.stats as stats
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.preprocessing import OneHotEncoder
from Milestone_2 import document_best_fit_pdf, document_pmf_data, performance_metrics, attack_correlation
import warnings
```

- These libraries are used for numerical computations, data manipulation, statistical analysis, machine learning, and performance evaluation.

3.2 Data Preprocessing

The dataset is loaded and split into training and testing sets:

```python
df = pd.read_csv("Train_data.csv")

training_df = df.iloc[:int(df.shape[0]*0.7),:]
training_attack = training_df['class']
testing_df = df.iloc[int(df.shape[0]*0.7):, :]
testing_attack = testing_df['class']

selected_df = df.iloc[:,0:41] #Selecting the columns without the class column
training_df_no_class = selected_df.iloc[:int(df.shape[0]*0.7),:]
testing_df_no_class = selected_df.iloc[int(df.shape[0]*0.7):, :]
```

- Columns without the target class are separated for further analysis and all the variables to be needed in the code are added to simplify reaching them.

3.3 Documenting Best-Fit Distributions

The function `document_analysis_results_ms3` calculates best-fit distributions for numerical and categorical data:

```python
def document_analysis_results_ms3(dict_1):
    df_1 = pd.DataFrame(dict_1)
    numerical_summary = document_best_fit_pdf(df_1)
    categorical_summary = document_pmf_data(df_1)
    return numerical_summary, categorical_summary
```

- Very small modifications were made in order to fix some errors in the input variables (i.e changing from a dictionary to a data frame).

3.4 Conditioned Data Preparation

```python
def conditioned_data(df_1):
    condition = df_1['class'].unique()
    df_1_no_class = df_1.copy()
    df_1_no_class = df_1_no_class.drop('class', axis=1)
    anomaly_conditioned_data = {}
    normal_conditioned_data = {}
    for column in df_1_no_class.columns:
        for value in condition:
            if value == 'anomaly':
                # Collecting the anomaly conditioned values together
                anomaly_conditioned_data[column] = df_1[df_1['class'] == value][column].dropna()
            else:
                # Collecting the normal conditioned data together
                normal_conditioned_data[column] = df_1[df_1['class'] == value][column].dropna()
    return anomaly_conditioned_data, normal_conditioned_data
```

- The function prepares the data by separating anomalies and normal cases into distinct datasets, which allows for tailored analysis using statistical fitting and modeling.

4 Task 1 code

4.1 Calculating PDF/PMF

```
def calculate_pdf_or_pmf(values, best_fit_params, is_categorical):
    if is_categorical:
        probabilities = best_fit_params.get('overall', best_fit_params)
        mapped_probs = values.map(lambda x: probabilities.get(x, 1e-6))
        return mapped_probs
    else:
        try:
            distribution_name = best_fit_params.get('best_fit_distribution')
            params = best_fit_params.get('params', {})
            if not distribution_name:
                raise ValueError("Missing 'distribution' key in best_fit_params.")
            distribution = getattr(stats, distribution_name)
            if isinstance(params, tuple):
                params = list(params)
            log_pdf = np.log(distribution.pdf(values, *params) + 1e-10)
            return log_pdf
        except Exception as e:
            print(f"Error calculating PDF: {e}")
            return None
```

- The function calculates the probability density function (PDF) for numerical data or probability mass function (PMF) for categorical data based on best-fit distribution parameters, handling various edge cases and errors gracefully.
- The log was applied to the values of the function since the pdf values were too small.

4.2 Naïve Bayes Anomaly Detection

The script calculates posterior probabilities using the Naïve Bayes theorem:

```
def naiive_bayes(df_res):
    print("\nCalculating Naive Bayes predictions...\n")
    def safe_calculate(col, fit_params, is_categorical=False):
        try:
            if fit_params and 'best_fit_distribution' in fit_params and fit_params['best_fit_distribution']:
                return calculate_pdf_or_pmf(df_res[col], fit_params, is_categorical)
            elif is_categorical and isinstance(fit_params, dict):
                return calculate_pdf_or_pmf(df_res[col], fit_params, is_categorical=True)
            else:
                return np.ones(len(df_res[col]))
        except Exception as e:
            print(f"Skipping column '{col}' due to error: {e}")
            return np.ones(len(df_res[col]))

    numerator_anomaly = num_numerator_anomaly * cat_numerator_anomaly

    numerator_normal = num_numerator_normal * cat_numerator_normal

    denominator = num_denominator * cat_denominator

    pr_normal_given_row = numerator_normal / denominator
    pr_anomaly_given_row = numerator_anomaly / denominator

    pr_normal_given_row = np.nan_to_num(pr_normal_given_row, nan=1e-10, posinf=1e10, neginf=1e-10)
    pr_anomaly_given_row = np.nan_to_num(pr_anomaly_given_row, nan=1e-10, posinf=1e10, neginf=1e-10)

    predicts = np.where(pr_anomaly_given_row > pr_normal_given_row, 'anomaly', 'normal')
    predictions_final = [1 if i == 'anomaly' else 0 for i in predicts]

    print(predictions_final)
    return predictions_final
```

- The posterior probabilities determine whether a record is classified as `'anomaly'` or `'normal'` using this formula:
  Pr(Anomaly—row) =
  $$\frac{\mathit{pdf}\, A|\text{Anomaly}(a) \cdot \mathit{pdf}\, B|\text{Anomaly}(b) \cdot \mathit{pdf}\, C|\text{Anomaly}(c) \cdot \ldots \cdot PMF Q|\text{Anomaly}(q) \cdot PMF R|\text{Ano}}{\mathit{pdf}\, A(a) \cdot \mathit{pdf}\, B(b) \cdot \ldots \cdot PMF Q(q) \cdot PMF R(r) \cdot \ldots}$$

- The function safe calculate was added to avoid or clearly display any exceptions happening during the calculations.

4.3 Results

The results of task 1 ...
Accuracy: 0.7417306165652289
Precision: 0.7123930444383108
Recall: 0.7393297049556001

5 Task 2 code

5.1 Encoding Categorical Features

Categorical columns are one-hot encoded for use in Naïve Bayes models:

```python
def encode_categorical_features(train_df_task_2, test_df_task_2):
    """
    Perform one-hot encoding for categorical features in train and test datasets using the same encoder.
    """
    categorical_columns = train_df_task_2.select_dtypes(include=['object']).columns
    categorical_columns = [col for col in categorical_columns if col != 'class']

    print("Encoding the following categorical columns:")
    for col in categorical_columns:
        print(f"- {col}")

    encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')
    encoded_train_data = encoder.fit_transform(train_df_task_2[categorical_columns])
    encoded_test_data = encoder.transform(test_df_task_2[categorical_columns])

    encoded_columns = encoder.get_feature_names_out(categorical_columns)

    encoded_train_df = pd.DataFrame(encoded_train_data, columns=encoded_columns, index=train_df_task_2.index)
    encoded_test_df = pd.DataFrame(encoded_test_data, columns=encoded_columns, index=test_df_task_2.index)

    train_df_task_2 = train_df_task_2.drop(columns=categorical_columns).join(encoded_train_df)
    test_df_task_2 = test_df_task_2.drop(columns=categorical_columns).join(encoded_test_df)

    return train_df_task_2, test_df_task_2
```

- This was done to transform them into a format suitable for Naïve Bayes models. This approach ensures that each category is represented as a distinct binary column, allowing the models to effectively incorporate categorical data into the prediction process.

5.2 Model Training and Evaluation

Three types of Naïve Bayes models are trained and evaluated:

```python
def train_and_evaluate_models(train_df, test_df):
    """
    Train and evaluate Gaussian, Multinomial, and Bernoulli Na ve Bayes models.
    """
    X_train = train_df.drop(columns=['class'])
    y_train = train_df['class']
    X_test = test_df.drop(columns=['class'])
    y_test = test_df['class']

    models = {
        'GaussianNB': GaussianNB(),
        'MultinomialNB': MultinomialNB(),
        'BernoulliNB': BernoulliNB()
    }

    for model_name, model in models.items():
        print(f"\nTraining {model_name}...")
        model.fit(X_train, y_train)
        predictions_task_2 = model.predict(X_test)

        accuracy = accuracy_score(y_test, predictions_task_2)
        precision = precision_score(y_test, predictions_task_2, pos_label='anomaly', zero_division=1)
        recall = recall_score(y_test, predictions_task_2, pos_label='anomaly', zero_division=1)

        print(f"Accuracy: {accuracy:.4f}")
        print(f"Precision: {precision:.4f}")
        print(f"Recall: {recall:.4f}")
```

- This function trains and evaluates three different Naïve Bayes models (Gaussian, Multinomial, and Bernoulli) on the provided training and test datasets, focusing on key classification metrics such as accuracy, precision, and recall.

- Each model is evaluated by computing performance metrics, which are essential for understanding how well the model handles the classification task, especially for detecting anomalies.

5.3 Results

```
Training GaussianNB...
Accuracy: 0.5585
Precision: 0.7704
Recall: 0.0725
```

- **Comment**: GaussianNB had moderate performance with a relatively high precision but low recall, suggesting it correctly identified most of the 'anomaly' class but missed many other cases.

```
Training MultinomialNB...
Accuracy: 0.5290
Precision: 0.4379
Recall: 0.0441
```

- **Comment**: MultinomialNB showed poor performance across all metrics, with very low recall and limited ability to capture anomalies, resulting in a high false negative rate.

```
Training BernoulliNB...
Accuracy: 0.9062
Precision: 0.9434
Recall: 0.8493
```

- **Comment**: BernoulliNB excelled because one-hot encoding transformed the categorical features into a sparse binary matrix, aligning perfectly with the model's design for binary data. This allowed BernoulliNB to efficiently calculate probabilities for each feature while handling the dataset's high dimensionality, leading to superior performance in anomaly detection.

6 Overview PCA MS 3 EXTRA

Principal Component Analysis (PCA) is a widely used dimensionality reduction technique in machine learning and data analysis. It transforms a set of possibly correlated variables into a set of uncorrelated variables known as principal components.

6.1 Key Concepts:

- PCA finds the directions (principal components) that capture the maximum variance in the data.
- The first principal component captures the highest variance, followed by the second principal component, and so on.
- The transformation reduces the dimensionality of the data while retaining as much information as possible.
- PCA can be applied to both numerical and categorical data.
- Common applications include noise reduction, feature selection, and visualization of high-dimensional data.

6.2 Steps in PCA:

1. Standardize the data.
2. Compute the covariance matrix.
3. Perform eigen-decomposition to find eigenvalues and eigenvectors.
4. Select the desired number of principal components based on explained variance.
5. Transform the data into the principal component space.

6.3 Usage in this milestone

1. The pre-made PCA functions from imported libraries were used.
2. We added a code to add a variable of PCA train and test values shifted to non-negative values.
3. All the columns became numerical after the application of PCA.
4. The data was compressed to only 11 columns instead of 41.
5. Tasks 1 and 2 were repeated for the new data after PCA.

7 general PCA code

7.1 Imports

```
import numpy as np
import pandas as pd
import scipy.stats as stats
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder,
    Binarizer, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB, MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, precision_score,
    recall_score, confusion_matrix, roc_curve
import warnings
from Milestone_2 import attack_correlation
```

- These libraries are used for numerical computations, data manipulation, statistical analysis, machine learning, and performance evaluation.

7.2 Data preparation

```
New_testing = pd.read_csv(’Test_data.csv’)
train_df_pca, New_testing_pca, train_df_pca_nonneg, New_testing_nonneg
    = apply_pca(train_df, New_testing, n_components=10)
train_bin_df, New_testing_bin_df = binarize_data(train_df_pca,
    New_testing_pca, threshold=0.0)
```

- This code snippet reads a test dataset from a CSV file. Then, Principal Component Analysis (PCA) is applied to reduce the dimensionality of the training and test data, ensuring that the most relevant features are retained. After applying PCA, the data is binarized using a specified threshold of 0.0 to convert continuous values into binary (0 or 1) representations.

- The new variable went through the same as variables in task 1 to prepare the data.

- Two PCA results were shifted to non-negative values to avoid errors in Multinomial Naïve Bayes estimation due to the negative values.

- The PCA results passed to the BernoulliNB were binarized to be suitable for it.

8 Key Differences in PCA version

8.1 Speed and computing

- The time taken to run the code and get results is a lot faster than the original data frame's time

- The best-fit distributions is done for 10 columns instead of 40 columns.

- More simple and shorter calculations are made consequently.

8.2 Predictions

```
predicts = np.where(
        (abs(pr_anomaly_given_row)*60 > abs(pr_normal_given_row)),
        'anomaly',
        'normal'
    )
```

- This condition is experimental and is based on printing and observing the data.
- The predictions resulting were very acceptable for Task 1 since task 2 is done elsewhere.

9 PCA Results

9.1 Task 1 results

Accuracy: 0.9013362127142664
Precision: 0.9506380120886501
Recall: 0.8289897510980966

- **Accuracy**: The accuracy improved significantly in PCA Task 1 compared to the regular Task 1. This substantial increase suggests that applying PCA helped in reducing noise and dimensionality, enhancing the model's ability to make better predictions.

- **Precision**: Precision saw a notable increase in PCA Task 1. This could be attributed to the dimensionality reduction and better feature extraction facilitated by PCA, leading to improved classification performance.

- **Recall**: Recall also improved significantly with PCA Task 1. By preserving the most important features through PCA, the model was better at identifying relevant data points, particularly in the cases of positive classifications.

9.2 Task 2 results

There is a noticeable overall improvement in the 3 performance parameters after applying PCA to our data.

Gaussian Naïve Bayes Performance...
Accuracy: 0.9143
Precision: 0.8779
Recall: 0.9455

- **Comment**: The PCA Task 2 results demonstrate noticeable improvements in both accuracy and precision, while maintaining a high recall. This highlights the effectiveness of PCA in reducing dimensionality and enhancing the model's ability to distinguish between normal and anomalous cases.

Multinomial Naïve Bayes Performance...
Accuracy: 0.8994
Precision: 0.9816
Recall: 0.7968

- **Comment**: The PCA application improves the precision and recall significantly while maintaining a high accuracy. The dimensionality reduction appears to enhance the Multinomial Naïve Bayes model's ability to capture meaningful data, resulting in better overall performance.

Bernoulli Naïve Bayes Performance...
Accuracy: 0.9306
Precision: 0.9473
Recall: 0.8996

- **Comment**: Bernoulli Naïve Bayes shows consistent performance across both Task 2 and PCA Task 2. The PCA might slightly improve the precision, but the recall and accuracy remain strong, suggesting that the model is robust even after feature extraction.

10 Challenges

1. **Small values after best fit pdf calculation:** After calculating the values of the best-fit probability density functions (pdfs) for each row, the resulting values were too small. When applying the Naive Bayes estimation, this led to a large number of `NaN` (Not a Number) and `Inf` (Infinity) values. To address this, we applied the `np.log` function to the values to prevent these numerical issues. Initially, we attempted multiplying by constants, but this did not resolve the problem, so we decided to continue using the `np.log` function to stabilize the calculations.

2. **Long running time before PCA:** The running time of the algorithm before applying Principal Component Analysis (PCA) was relatively long due to the high dimensionality of the dataset. However, after applying PCA, the number of columns in the dataset was reduced from 41 to 11, resulting in a fourfold improvement in processing speed.

3. **Choosing an appropriate threshold for predictions:** We faced difficulties in determining a suitable threshold for making predictions. Initially, we experimented with several thresholds before PCA, but the results were consistently average, with accuracy in the 70s range. After applying PCA and fine-tuning the threshold through trial and error, we were able to achieve better final results with a more appropriate threshold.

4. **Negative values in Multinomial Naive Bayes estimation:** The Multinomial Naive Bayes estimation requires non-negative data to function correctly. However, the normal PCA process resulted in negative values, which posed a challenge. To overcome this, we applied a shift to the PCA-transformed data, ensuring all values became positive, which allowed the Multinomial Naive Bayes model to work as expected.

5. **Binarizing data for Bernoulli Naive Bayes estimation:** The Bernoulli Naive Bayes estimation requires binary data (0s and 1s) for optimal performance. However, the PCA-transformed data initially contained continuous values. To resolve this, we binarized the data before applying the Bernoulli Naive Bayes estimation, which significantly improved the model's performance and ensured it generated better results.

11 Conclusion

- This project successfully demonstrated the effectiveness of Principal Component Analysis (PCA) and Bernoulli Naïve Bayes (BernoulliNB) in the context of anomaly detection within intrusion detection datasets.

- Principal Component Analysis (PCA): PCA proved to be a valuable preprocessing technique for dimensionality reduction, reducing the number of features from 41 to 11 while preserving most of the information in the data. This compression significantly improved computation efficiency and reduced noise, leading to enhanced model performance. By simplifying the feature space, PCA enabled faster training and testing cycles while maintaining or improving predictive capabilities.

- Bernoulli Naïve Bayes (BernoulliNB): BernoulliNB emerged as the most effective classifier in this study, achieving outstanding results in accuracy, precision, and recall. Its suitability for binary features and ability to handle the transformed dataset effectively made it the ideal choice for this application. When applied to the PCA-reduced dataset, BernoulliNB further capitalized on the cleaner, lower-dimensional feature space, solidifying its robustness and reliability.

- In conclusion, the combination of PCA for preprocessing and BernoulliNB for classification provided a powerful pipeline for anomaly detection, achieving high performance while optimizing computational resources. This approach is particularly advantageous for large-scale intrusion detection systems requiring efficiency and precision.

12 References

1. Principal Component Analysis (PCA). Retrieved from `https://www.geeksforgeeks.org/principal-component-analysis-pca/`

2. IBM. Principal Component Analysis. Retrieved from `https://www.ibm.com/think/topics/principal-component-analysis`

3. Stack Overflow. Handling NaN with log in Pandas. Retrieved from `https://stackoverflow.com/questions/57696132/pandas-nan-with-log`