CS 584 (Theory and Applications of Data Mining)
Fall 2024
HW 02

Md Sultanul Islam Ovi
G01444040
movi@gmu.edu

# Report: Drug Activity Prediction

**Miner Username: movi**
**F1 Score: 0.85**
**Rank: 1**

## 1 Introduction

This homework aims to classify compounds as active (**1**) or inactive (**0**) using **Decision Tree** and **Naïve Bayes** models. The focus was on handling imbalanced data with effective feature selection using **SelectKBest**. This report outlines the data preparation, model implementation, parameter tuning, and results achieved on the leaderboard.

## 2 Data Preprocessing

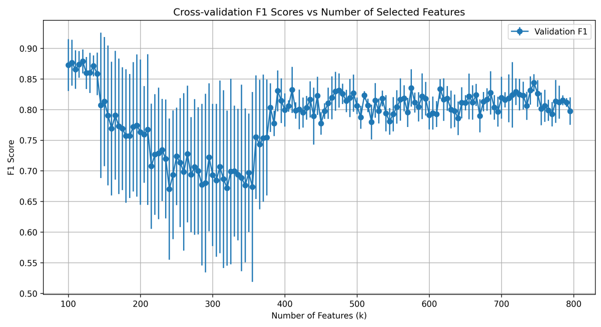
The input data was provided as a sparse binary matrix, with non-zero feature indices marked as **1** and all other entries set to **0**. Both training and test datasets were converted into sparse matrices to ensure consistency during model training and evaluation.

To address class imbalance in the dataset, different sampling techniques were applied **individually** in separate experiments. **SMOTE** was used to generate synthetic samples for the minority class, aiming to balance the dataset by oversampling. In another experiment, **SMOTEENN** was applied, which combined SMOTE with Edited Nearest Neighbors to oversample and remove noisy data points. **Random undersampling** was also tried in separate runs, where the majority class was reduced in size to match the minority class, though this approach risked losing valuable information.

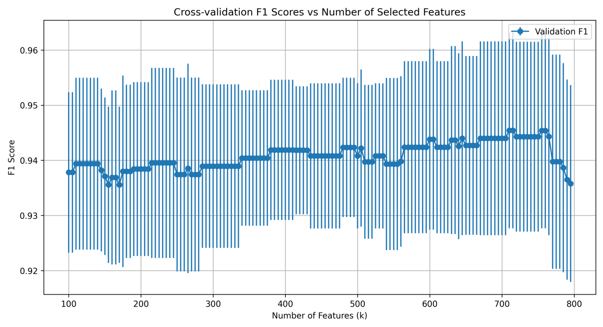
Feature selection and dimensionality reduction were applied to enhance model efficiency. **SelectKBest** with **chi-square** was used to select the top $k$ most relevant features for training. Other experiments were conducted where **Truncated SVD** was applied on top of **SelectKBest** to further reduce dimensionality while preserving as much variance as possible.

Two models were implemented for classification: **Decision Tree** and **Naïve Bayes**. For **Decision Tree**, various **class weight configurations** were explored to manage class imbalance and improve model generalization. The tree's **depth** and **number of nodes** were tracked to monitor and prevent overfitting. For **Naïve Bayes**, **Bernoulli Naïve Bayes** was used along with class balancing techniques to optimize performance.

**Stratified K-Fold Cross-Validation** with **5 splits** was employed to evaluate the models consistently. This ensured that the class distribution remained balanced across both training and validation splits, reducing bias and improving the robustness of the results.



(a) Decision Tree F1 Scores vs Number of Selected Features



(b) Naïve Bayes F1 Scores vs Number of Selected Features

Figure 1: Comparison of Cross-Validation F1 Scores for Decision Tree and Naïve Bayes Classifiers

During training, detailed **precision**, **recall**, and **F1-scores** were calculated for both classes (Class **0** and Class **1**). The **average F1-score** was used as the primary metric to compare configurations and select the top-performing models. The analysis of cross-validation F1 scores for both Decision Tree and Naïve Bayes models, as shown in Figure 1, helped identify clusters of optimal values for $k$. These clusters were used to guide the selection of $k$ values for training and testing the final models, ensuring better performance and avoiding overfitting.

## 3 Experiments and Results

I performed several experiments to find the best-performing configuration by tuning parameters, testing classifiers, and adjusting sampling strategies. Both **Decision Tree** and **Naïve Bayes** classifiers were evaluated with different class weights and sampling methods to identify optimal setups.

To achieve the best results, I used cross-validation to explore combinations of feature selection techniques and sampling strategies. **SelectKBest** with chi-square was applied for feature selection, and in some cases, **Truncated SVD** was used on top of SelectKBest.

The models were trained and tested on both the **GMU Hopper Cluster** and the **Lightning AI platform**. Each extensive parameter search took around **8 to 10 hours** to complete. After selecting the best classifier, sampling strategy, feature selection method, and $k$-value, the final models were built. These models were ranked based on performance metrics such as precision, recall, and F1-scores. The optimized model runs efficiently, generating predictions in under **10 seconds**.

Table 1: Comparison of Different Tests and Results

| **Count** | **Classifier** | **Weights** | **Sampling** | **Feature Selection** | **K Value** | **Training F1** | **Leaderboard F1** |
|---|---|---|---|---|---|---|---|
| 1 | Naïve Bayes | Uniform | SMOTE | SelectKBest | 613 | 0.94 | **0.85** |
| 2 | Naïve Bayes | Uniform | SMOTE | SelectKBest | 707 | 0.93 | 0.83 |
| 3 | Decision Tree | {0.8, 0.2} | Random undersampling | Truncated SVD | 370 | 0.93 | 0.70 |
| 4 | Naïve Bayes | Uniform | SMOTE | SelectKBest | 595 | 0.91 | 0.81 |
| 5 | Naïve Bayes | {0.9, 0.1} | SMOTEENN | SelectKBest | 20 | 0.89 | 0.64 |

## 4 Result Summary

The final model achieved a training F1-score of **0.9424** and a leaderboard F1-score of **0.85**, ranking **1st** on the leaderboard. The top-performing model used the **Bernoulli Naïve Bayes** classifier with **SelectKBest** feature selection, retaining the top **613** features.

## 5 Instructions for Running the Code

To reproduce the results, follow these steps:

1. Install the necessary libraries.
2. Run the script `drug_prediction_classifier.py`.
3. Ensure that the dataset files `train.dat` and `test.dat` are in the same directory as the script.

Additionally, an `ipynb` notebook named `drug_prediction_parameter_search.ipynb` is provided. It contains 6 extra sections dedicated to an extensive search for optimal parameters and feature selection methods.