Company Bankruptcy Prediction

Objective : Develop a machine learning model to predict whether a company will go bankrupt based on financial indicators of the company.

OUTLINE :

1. Exploratory Data Analysis
2. Data Preprocessing
    - Feature Extraction and Over Sampling
3. Model Architecture
4. Evaluation
    - Metrics : Accuracy, Precision, Recall, F1 Score

Exploratory Data Analysis

Exploring Patterns in Data

- Identified the null values and duplicate rows in dataset (None).
- **Dataset : 5455** rows x **96** columns **(5455** datapoints, **95** features and **1** target variable)
- **Class Distribution and Imbalance:**
    - Number of Bankrupted companies (1) : **154**
    - Number of Non-Bankrupted companies (0): **5301**

Handling Redundant Data and Errors

- **Feature Type Identification :** Out of 95 features, 93 were Numerical Features and 2 were Categorical Features. (‘**Liability-Assets Flag**’ and ‘**Net Income Flag**’). Since, ‘**Net Income Flag**’ had the same value (1) for all datapoints, it was dropped.
- **Data Error Handling:** Detected features with extremely high mean values caused by false data in some rows (instead of between 0 and 1) .
    - **Dropped Features:** 9 features with >800 errors were removed.
    - **Median Replacement:** 14 features with <200 errors were corrected.
    - **Final Features:** 86 retained.

Feature Correlation

- **Pairwise Correlation :**

Calculated the absolute correlation between all feature pairs. If any pair had a correlation greater than **threshold** value **0.90**, one feature was removed.

- **Target Correlation :**

For each correlated pair, we kept the feature with a higher absolute correlation to the target (‘Bankrupt?’) and dropped the other. This **reduced** the number of **features from 86 to 62**.

Data Preprocessing

Feature Extraction Using ANOVA

- **ANOVA** or **Analysis of Variance** is used to determine whether a feature is significant enough to be used for training or not.
- This is done by calculating **F-scores** for each feature. A group is a class of the target variable. If the F-score is high for a feature, it means that the values for this feature vary highly between the target variable classes and vary less within a single class. This makes the feature significant.

$$F = \frac{\text{Variance between groups}}{\text{Variance within groups}}$$

- The variance between groups for 2 classes is given by:

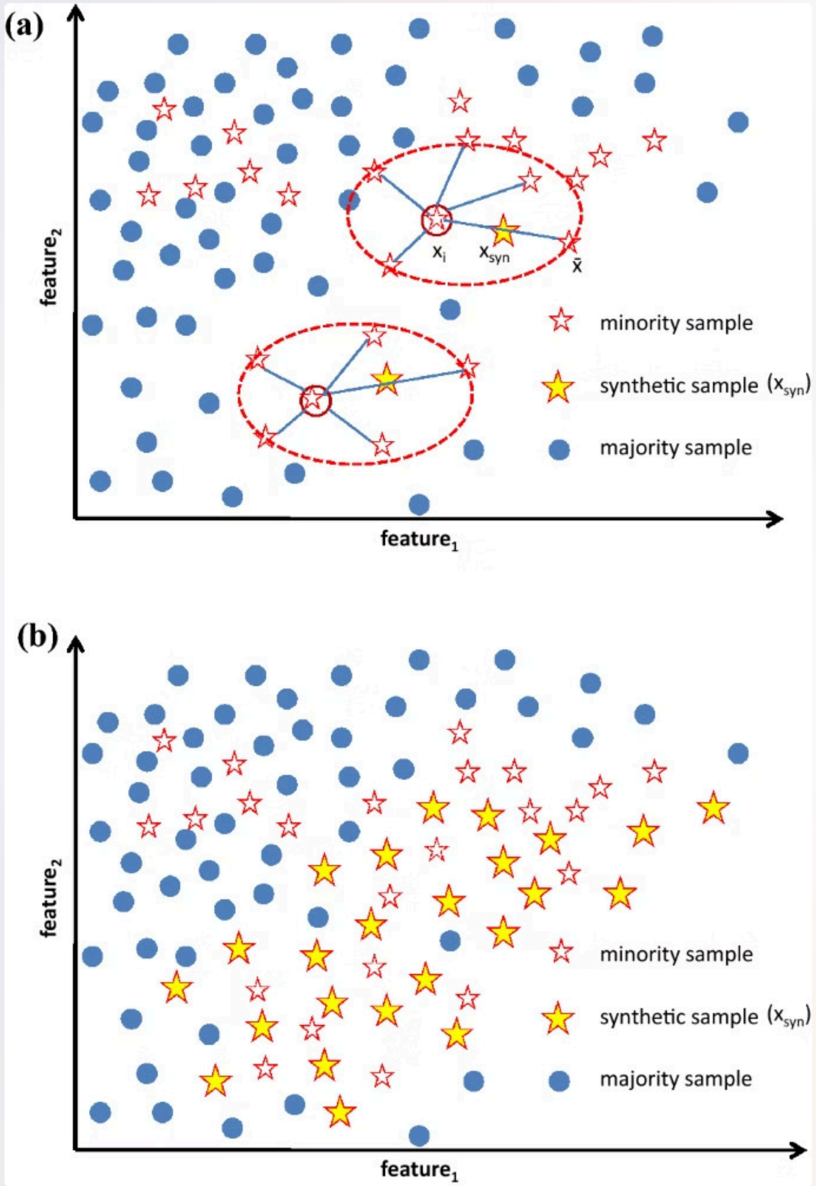
$$S_B^2 = \sum_{i=1}^{k} N_i(\mu_i - \mu)^2$$

  - Where k = 2 (number of classes)
  - Ni represents number of datapoints in the class
  - mu is the mean of all the datapoints of that feature and mu(i) is the mean of that feature in the class
- The variance within groups is calculated by summing the variances in each class weighted by the sample size of that class. Comparing the between-group variance to within-group variance provides a measure of the feature's significance.

```
def select_features_anova(X_data, y_data, top_n=30):

    selector = SelectKBest(score_func=f_classif, k=top_n)
    selector.fit(X_data, y_data)
    mask = selector.get_support()
    selected_features = X_data.columns[mask].tolist()
    return selected_features
```

- Here SelectKBest selects the **best 30 features** based on their F-scores and mask takes in the returned boolean array from get_support() (True is the feature is selected and false otherwise). selected_features extracts only the selected features from the mask.



Oversampling using SMOTE

When a dataset is imbalanced models tend to be biased toward the majority class. SMOTE addresses this by artificially increasing the number of minority class samples. In our case initially we had **154** of class '1' and **5301** of class '0'.

After the 80-20 train-test split, we had **4241** of class '0' and **123** of class '1' in our train data.

Selects a class sample randomly and finds its k nearest neighbours.

Selects one of the neighbours and generates a new sample between it and the original sample using linear interpolation.

After using SMOTE we have 4241 samples each for both classes.

We do not apply SMOTE on the test data.

Standardisation

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_sm)
X_test_scaled = scaler.transform(X_test)
```

We use the **StandardScaler** that makes all feature values have 0 mean and unit variance to avoid giving unnecessarily large weights to large-scale features.

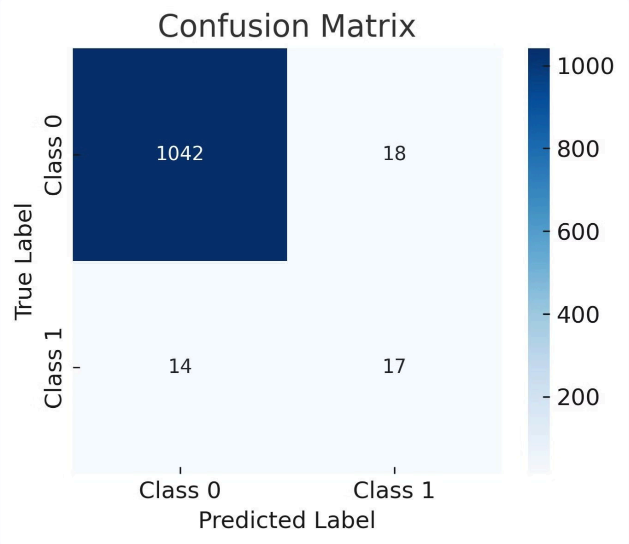
Model Architecture

```
def create_dnn_model(input_dim):
gnb_model = GaussianNB()
```

- This function takes in input as number of features. The DNN has an **input layer, 3 hidden layers and output layer.**
- Input layer takes features and input.
- **First Hidden Layer**: 256 neurons, uses **ReLU activation function**, **Batch Normalization**, **Dropout** (0.5) (drops 50% neurons randomly during training)
- **Second Hidden Layer**: fully connected 128 neurons, uses ReLU activation function, Batch Normalization and Dropout (0.5).
- **Third Hidden Layer** : fully connected 64 neurons, uses ReLU activation function, Batch Normalization and Dropout (0.4).
- **Output Layer**: fully connected with 1 neuron, uses sigmoid activation for binary classification.
- **Optimizer** : Adam (Adaptive Moment Estimation) (learning rate =0.0005)
- **Loss Function** : Binary Crossentropy (log-loss)
- We train the DNN model for **200 epochs**, **batch size 64** (number of training samples processed before weights are updated) and **20% validation split**. We get predicted probabilities from each model and combine the probabilities by averaging them.
- The Gaussian Naive Bayes model (GNB) model applies **Bayes' theorem**, assuming that features are **independent** given the class label. It calculates the probability of bankruptcy by multiplying the **Gaussian likelihoods** of each feature and normalizing using prior probabilities.
- We implement an ensemble approach using **soft voting**, where the final probability is obtained by averaging the predicted probabilities from both models.
- Then we tune the **threshold** that decides the value of the average probability above which the model predicts bankruptcy. This is done by calculation of f1-score for each threshold value.

Evaluation

Using our "Train.csv" dataset, we evaluated on the 20% test data split and obtained the following results for various metrics:



Results :

- **Precision**: 0.4857
- **Recall**: 0.5484
- **F1-Score**: 0.5152
- **Accuracy :** 0.9723
- **Best threshold**: 0.45