Contents

Introduction:

The report objective is to help the bank in developing a classification model to predict if the client will subscribe to a 'term deposit' or not. This prediction will be very essential to the bank to optimise its marketing strategies, improve customer targets, increase or reduce focus on certain campaigns. The report outlines the processes of data preparation, model selection, hyperparameter tuning, evaluation metrics, and results analysis. Random Forest and Support Vector Classifier (SVC) are trained, tested and evaluated to determine the best-performing model for deployment.

Data Preparation

Data Preparation: Helps to create a dataset that is clean, consistent and is balanced for modelling. A properly balanced and prepared dataset improves model performance. The process involves the following:

1. Reading
2. Cleaning
3. Encoding
4. Scaling
5. Splitting
6. Balancing

The dataset 'bank.csv' is mounted using '*pandas.read_csv()*' which has 4521 rows and 17 columns. It includes both numerical and categorical features such as age, job, marital status, education, and more.

1. Reading the data

Load the data using 'read_csv' . The dataset contains 4521entroes/rows and 17 columns/features. It includes both numerical and categorical variables. The target variable 'y' is the main column that determines the approach.

```
> data = read_csv('/content/drive/MyDrive/ML_Stats/bank.csv')
> data.info()
```

2. Missing data

We do a missing value check to make sure that the data is consistent and balanced. No missing values are found in this dataset indicating a clean dataset.

```
> print(data.isnull().sum())
```

3. Data Encoding

For machine to learn the data, it should be in a format that is readable by the machine. So any models/machines can't process the categorical data directly, as they cant understand what the data is.

To represent the categorical data without any bias, we would segregate the data into 2 separate encoding. One is called 'Binary encoding' and other is 'One-hot encoding'

3.1 Binary encoding

Selecting all the categorical data columns that have any 2 entries. Converting categorical variables like 'default', 'housing', 'loan', and the target variable 'y' into binary values (0 & 1)

```
> data['default'] = data['default'].map({'yes':1, 'no':0})
> data['housing'] = data['housing'].map({'yes':1, 'no':0})
> data['loan'] = data['loan'].map({'yes':1, 'no':0})
> data['y'] = data['y'].map({'yes':1, 'no':0})
```

3.2 One-hot encoding or get_dummies

Here the categorical variables are encoded as numerical, where a separate column is added for each unique category of each feature. 'get_dummies' is used to create dummy columns for each unique feature. The more categories a feature has, the more columns are created, increasing memory usage and computational complexity. It is generally used for variables that can't be ranked.

> data2 = get_dummies(data, columns = ['job', 'marital'], dtype=int)

3.3 Label encoding – Ranking

One of the main reasons to use 'Label Encoding' instead of One-Hot Encoding, is to mitigate the **'Curse of Dimensionality'**, which in-turn leads the model to overfit.

Label Encoding converts categorical values into numerical values by assigning a unique number to each category. This is mainly used when the categories under the variables can be ranked or has a meaningful order, converting it to an ordinal data. This encoding allows the machine to process the hierarchal nature of the data easily and to avoid the 'Curse of Dimensionality' and overfitting that can be caused by one-hot encoding.

> data['education'] = data['education'].map({'primary':1,'secondary':2,'tertiary':3, 'unknown':4})
> data['contact'] = data['contact'].map({'cellular':1,'unknown':2,'telephone':3})
> data['month'] =
data['month'].map({'jan':1,'feb':2,'mar':3,'apr':4,'may':5,'jun':6,'jul':7,'aug':8,'sep':9,'oct':10,'nov':11,'dec':12})
> data['poutcome'] = data['poutcome'].map({'unknown':1,'failure':2,'other':3,'success':4})

4. Data Scaling

Before scaling the dataset, we would drop the target variable, 'y' and assign it to a different data frame.

> X = data2.drop('y', axis = 1)
> Y = data2['y']

Scaling is necessary as we require all the categorical variables are on a similar scale. If this step is not performed several issues can occur, like leading to biased results or poor model performance.

We scale the dataset using '*> X_scaled = StandardScaler().fit_transform(X)*' by importing 'StandardScaler()' from sklearn library.

5. Data Splitting

Once the data is scaled, we split the data into training and testing sets. We import 'train_test_split' from sklearn library, where we split the data into a 70:30 ratio.

Data splitting is a crucial step in machine learning that involves dividing the dataset into training and testing sets. In a 70:30 split,

- 70% is used for **training** the model, helping it learns patterns from the data.
- 30% is used for **testing**, evaluating how well the model generalizes to unseen data.

This ensures that the model is trained on a substantial portion of the data while leaving enough for unbiased evaluation

6. Data Balancing

When dealing with imbalanced datasets, where minority class(here: clients who subscribed to a term deposit) has fewer samples than majority class, which might create a bias towards the majority class.

We use SMOTE (Synthetic Minority Over-sampling Technique) to create a balanced dataset, which in-turn generates synthetic instances by interpolating between 2 samples creating a new sample.

> X_train, Y_train = SMOTE (random_state = 61).fit_resample(X_train,Y_train)

Model Hyperparameter Tuning

Hyperparameters are critical for optimizing the model performance. As these are unknown as for what is the best numbers to use it for, we need to use different methods(such as method 2 and 3) to let the machine tell us which is the best value for the hyper parameters.

1. For Random Forest Classification:

Hyperparameters tuned is: "**n_estimators**"

- To build a random forest classifier model with the best possible 'forests', we need to tune n_estimators using GridSearchCV.
- For this model, the bank wishes to see all the positive subscribers, so we will use the scoring method as 'recall'.

1.1 How to tune the hyperparameters:

We use GridSearchCV to determine the hyperparameters

> no_trees = {'n_estimators': [50, 100, 200, 250, 300, 350, 400, 450]}
> RF_grid_search = GridSearchCV(estimator=RF_classifier2, param_grid=no_trees, scoring='recall', cv=5)

Reason to tune the hyperparameters:

- Too less trees would result the model into Underfitting.
- Too many trees may overfit and increases the Time, operational cost and decrease the model efficiency.

Hyperparameters that can be tuned are:

- max_depth = Limits tree depth to prevent overfitting
- min_samples_split = Sets the minimum samples required to split a node.

Result:

Tuning this hyperparameter (n_estimators) – gives the optimal output of 'n_estimators = 50' and 'recall score of 23.8%'.

2. For Support Vector Machine (SVM):

Hyperparameters tuned are: **"kernel" and "c"**

"c" or Regularization Parameter = balances the margin width.

- Small value of c – helps prevent overfitting as the model has higher bias but lower variance.
- Large value of c – risks overfitting but makes the model fit the training data more closely.
- Default range of c is [.001,.01,.1,1,10,100]

"kernel" = defines the decision boundary

- Kernel transforms input data into a higher-dimensional space to make it easier to find a decision boundary.
- This is particularly useful for handling non-linearly separable data.
- There are 4 types of kernels ['linear', 'poly', 'rbf', 'sigmoid']

2.1 How to tune the hyperparameters:

We use GridSearchCV to determine the hyperparameters.

> kernels_c = {'classification__kernel': ['linear','poly','rbf','sigmoid'], 'classification__C': [.001,.01,.1,1,10,100]}
> SV_grid_search = GridSearchCV(estimator = SV_classifier2, param_grid=kernels_c, scoring='recall', cv=5)

Result:

Tuning the hyperparameters 'c' = 0.001 and 'kernel' = rbf achieved an accuracy of 79.08%.

Choice of Evaluation Metrics:

The choice to select the evaluation metric while initializing the GridSearchCV is the most important factor, as this decides which feature to focus on.

As per the task given where, "the bank wants to use a classification model that can predict whether the client has subscribed a term deposit?" – **showcases the importance of 'recall'**

Analysis of Matrix

Given that the bank's primary goal is to predict whether the client has subscribed a term deposit (focussed on recall, the Support Vector model would be the better choice for deployment despite its lower accuracy, precision, and F1 score compared to Random Forest.

1. **Accuracy and Precision**: While the Random Forest model has higher accuracy (90.13%) and precision (60.55%) compared to Support Vectors accuracy(65.14%) and precision(22.38%), these metrics are less critical for the bank's goal of identifying as many subscribers as possible. The lower precision of the Support Vector model (22.38%) indicates that it may produce more false positives, but this is ok as the priority of the bank is to capture as many true positives as possible.
2. **Recall**: The Support Vector model has a significantly higher recall (81.53%) compared to the Random Forest model (42.04%). This means that the Support Vector model is much more effective at identifying clients who have actually subscribed to a term deposit, which aligns with the bank's objective.
3. **F1 Score**: The F1 score for the Support Vector model (0.3512) is lower than that of the Random Forest model (0.4962), but again, this is secondary to the bank's focus on recall.

Final Analysis:

Deploy the Support Vector Model: Since the bank prioritizes recall(if client has subscribed a term deposit), the Support Vector model is the more suitable choice for deployment. It will help the bank identify a larger number of clients who are likely to subscribe to term deposits, which is the primary goal of the classification task and as nothing else was taken into consideration.

Overfitting avoidance mechanism

Two main techniques were used for controlling overfitting: Feature Selection and Regularization. Both the models are used collectively to reduce the overfitting.

1. Curse of Dimensionality**: One-hot encoding** is used to convert the categorical variables ('job' & 'marital') into binary format. But this can significantly increase the number of features which will definitely result in over-fitting. Taking one step further, we use **Label encoding**, where the variables ('education', ' contact', 'month', 'poutcome') are ranked. As Label encoding reduces additional dimensions that can be caused by one-hot encoding, we **avoid the curse and prevent overfitting** by using both Label encoding (for variables that can be ranked) and One-hot encoding(for variables that cannot be ranked).
2. Cross Validation: GridSearchCV is used for hyperparameter tuning(n_estimators), which splits data into 5-fold cross validation, training, testing, evaluating and ranking for the best performance across multiple folds. It helps to avoid overfitting by ensuring that the model's performance is consistent.
3. Regularization (c): Small value of c, increases the regularization strength, which results in a larger margin and a simpler model. This helps prevent overfitting but may lead to underfitting if the value is too small. A large value of C reduces the regularization strength, allowing the model to fit the training data more closely, but this increases the risk of overfitting. By tuning C, the model achieves a balance between fitting the training data and generalizing to unseen data

Recommendation for Deployment:

The outputs of model Random Forest and Support Vector after using several techniques to avoid overfitting, reducing dimensions etc are shown below:

Matrix	Random Forest	Support Vector
Accuracy	90.13%	65.14%
Precision	60.55%	22.38%
Recall	42.04%	81.53%
F1	49.62%	32.12%
Conf. Matrix TP	43	128
TN	1157	756
FN	91	29
FP	43	444

Scenario 1:

Based on the performance metrics, its recommended to deploy Ransom Forest model. Its accuracy of 90.13% is higher than the Support Vectors' 65.14%. While the SVC has a higher recall, its low precision indicates that it may incorrectly predict many clients as subscribers, which could lead to unnecessary marketing efforts and costs for the bank. The Random Forest model provides a more reliable prediction, making it a better choice for real-world application.

Scenario 2:

Based on the absolute requirement mentioned in the task while nothing else is considered, where 'The bank wants to use a classification model that can predict whether the client has subscribed a term deposit' – it is recommended to deploy Support Vector model. This is focused on high recall, But we need to be cautious as with a high recall and low precision it may incorrectly predict many clients as subscribers even when they are not.

Underfitting Analysis

1. **Random Forest:** This model offers a good balance between precision and recall. It even offers a high accuracy but is not so efficient in identifying true positives. While, this model has high accuracy(90.13%), it has a relatively low recall(42.04%) which is required to this query. So, I believe that there might be some level of underfitting here as it doesn't seem to capture TP cases effectively.
2. **Support Vector:** This model shows signs of underfitting due to its low accuracy (65.14% compared to RF: 90.13%) and f1 score (32.12% compared to RF: 49.62%). SVC is a sensitive data and is affected by many factors.

Possible Reasons for Underfitting in SVC

- Complex Model: It seems that out model is not complex enough to deliver a solid performance, as the value of c(c=0.001) might be very low – which makes the model simple and less complex.
- Even though the data is scaled, SVC is sensitive to scaling. Also, improper scaling can lead to underfitting.
- Data imbalances: Even though SMOTE is applied to balance the data, the model might still face issues with the highly imbalanced nature of target variable 'y'.