DA3 - Assignment 3 | GitHub

Technical Report: Predictive Modeling of Defaulted Firms in the Computer, Electronic, and Optical Products Industry

By: Dávid Szabados & Hla Myitzu

Executive Summary

This technical report outlines the process and findings from the task of building predictive models to classify defaulted firms in the 'Manufacture of computer, electronic and optical products' industry in 2015. Our objective is to predict defaults based on the data from 2014, with a focus on small and medium enterprises (SMEs).

Introduction

The notebook begins with the setup of Python environment necessary for conducting data analysis and modeling. It includes the importation of various libraries and frameworks essential for statistical analysis, machine learning, data visualization, and preprocessing. Notably, it incorporates:

1. **Library Imports**: The first cell imports numerous libraries that are essential for data analysis, visualization, and machine learning. This includes `pandas` for data manipulation, `seaborn` and `matplotlib` for plotting, `statsmodels` and `sklearn` for statistical modeling and machine learning, among others. It suggests that the notebook will cover comprehensive data analysis and modeling tasks.

2. **Path and Helper Functions**: The second cell sets up the current working directory and appends a "utils" folder to the system path, likely to import custom helper functions defined in an external script (`py_helper_functions`), indicating that the notebook might use custom utility functions for specific tasks.

3. **Data Section (Markdown)**: This cell, marked by the header "DATA," suggests that the following section will deal with data handling.

4. **Data Importation and Preprocessing**: This cell reads data from a CSV file hosted on GitHub, performs some filtering, and carries out an operation to fill missing combinations of 'year' and 'comp_id' with NaNs. This implies a focus on preparing a panel data structure for analysis.

5. **Label Engineering (Markdown)**: This heading indicates that the subsequent section might involve creating or modifying labels (target variables) for machine learning models.

The notebook configures the Python environment by suppressing warnings and adding a custom utilities directory to the system path for additional helper functions. This indicates the use of external Python scripts (py_helper_functions) to streamline analysis or preprocessing tasks.

Data Section

A section titled "DATA" suggests the notebook involves data loading, exploration, or preprocessing tasks.

The notebook is structured around a comprehensive data analysis project, including:

- Preprocessing and cleaning of data.
- Exploratory data analysis (EDA) to understand the dataset's characteristics.
- Application of statistical models to infer relationships between variables.
- Use of machine learning models for prediction or classification tasks.
- Evaluation of model performance and selection of the best-performing models.

Label & Feature Engineering

The focus is on further preprocessing of the data and the initial steps towards feature engineering. The steps covered include:

1. **Creating Status Indicators for Companies**: New variable, `status_alive` is created to indicate whether a company was "alive" (had positive sales and not missing sales data) in the given year. This involves conditional operations based on the `sales` and `year` columns, showcasing an initial step in label engineering.

2. **Defining Company Default**: A new variable, `default`, is defined to identify companies that existed in a certain year but did not exist in the following one, based on the previously created status indicator. This variable serves as a potential target for predictive modeling, focusing on predicting company defaults.

3. **Filtering Data by Year**: The dataset is filtered to include only some of the years, before the time of the holdout set, from 2010 up to 2013, to serve a basic time frame for the training set.

4. **Filtering by Sales**: Sales data is further filtered to include only entries with sales less than 10 million, likely to focus on a specific size of companies or to remove outliers that could skew the analysis.

5. **Handling Negative Sales Values**: Negative sales values are replaced with 1. This adjustment could be aimed at facilitating log transformations or ensuring that all sales values are non-negative for subsequent analysis.

6. **Creating Log-transformed and Scaled Sales Features**: New features are created, including the natural logarithm of sales (`ln_sales`), sales in millions (`sales_mil`), and the log of sales in millions (`sales_mil_log`). Log transformations are commonly used to normalize the distribution of variables and manage skewed data, while scaling by millions could be relevant for ease of interpretation or analysis.

7. **Calculating Year-over-Year Change in Log-transformed Sales**: A variable (`d1_sales_mil_log`) is created to capture the year-over-year change in the log-transformed sales in millions. This feature could be useful for modeling growth or contraction of company sales over time.

8. **Adjusting Company Age and Indicating New Firms**: The `age` of companies is adjusted to account for new firms, with a specific dummy variable (`new`) created to flag these instances. This includes handling cases where the `age` calculation might result in negative values (indicating data inconsistencies or firms founded within the year of observation) and identifying firms with a balance sheet not covering a full year.

9. **Adjusting Sales Growth for New Firms**: The notebook adjusts the previously calculated year-over-year sales growth (`d1_sales_mil_log`) for new firms, setting it to 0 for these firms to account for their lack of historical sales data. This step ensures that the variable is appropriately handled for all companies, including those newly founded.

10. **Industry Category Dropping**: The `ind2` variable, which represents industry codes, is dropped entirely from the dataset, as we are only looking for those companies, with an ind2 code of 26, Manufacture of computer, electronic and optical products, thus keeping companies outside of this industry is unnecessary and biases the modeling.

11. **Creating Firm Characteristics Variables**: New variables are created to capture firm-specific characteristics more accurately:
    - Squared age (`age2`) to potentially model nonlinear effects of company age on outcomes.
    - A binary indicator for foreign management (`foreign_management`), based on the proportion of foreign ownership.
    - Categorical transformations for `gender` and regional location (`m_region_loc`), likely to prepare these variables for use in statistical models that can handle categorical inputs.

12. **Identifying Financial Anomalies**: A flag (`flag_asset_problem`) is introduced for companies with negative or missing values in their asset-related variables (`intang_assets`, `curr_assets`, `fixed_assets`). This flag could be crucial for identifying and handling potential data quality issues or financial distress signals.

13. **Adjusting Asset Variables**: Asset-related variables (`intang_assets`, `curr_assets`, `fixed_assets`) are adjusted to ensure no negative values, which could be critical for financial ratio calculations and overall data integrity.

14. **Calculating Total Assets**: A new variable, `total_assets_bs`, is generated by summing up the adjusted asset variables. This is a fundamental step in financial analysis, providing a base for numerous ratio calculations.

15. **Normalizing Financial Variables**: The notebook normalizes various profit and loss (`pl_names`) and balance sheet (`bs_names`) items by sales and total assets, respectively. This standardization creates new columns for each item, facilitating comparison across companies of different sizes and industries.

16. **Handling Extreme Values**: The notebook introduces flags for potentially problematic financial ratios (e.g., values that are unrealistically high or low) and normalizes certain ratios to ensure they fall within a plausible range. This includes setting high values to 1, indicating a potential data quality issue or outlier, and adjusting variables that could theoretically span a wide range but are typically constrained within specific bounds.

17. **Dropping Flags with No Variation**: The notebook concludes this segment by dropping flag variables that show no variation, streamlining the dataset for analysis or modeling by removing redundant information.

18. **Handling Missing Values**: Several columns (`COGS`, `finished_prod`, `net_dom_sales`, `net_exp_sales`, `wages` for example) contain a high number of missing values. Variables with more meaning got their missing values filled with the median of its column. Categorical missing values are filled with “Missing”, after creating it as a new category, for this reason. At last, the remaining variables, which contain float or integer values, are filled with 0 and rounded to 3 digit, to avoid infinity values.

Cross Validation

```
logit_models = dict()
CV_RMSE_folds = dict()

for i, model_vars in enumerate(logit_model_vars):

    model_equation = "default~" + "+".join(model_vars)
    y_train, X_train = patsy.dmatrices(model_equation, data_train)

    LRCV_brier = LogisticRegressionCV(
        Cs=C_value_logit,
        cv=k,
        refit=True,
        scoring="neg_brier_score",
        solver="newton-cg",
        tol=1e-7,
        random_state=42,
    )

    logit_models["M" + str(i + 1)] = LRCV_brier.fit(X_train, y_train)

    # Calculate RMSE on test for each fold
    CV_RMSE_folds["M" + str(i + 1)] = np.sqrt(
        -1 * logit_models["M" + str(i + 1)].scores_[1].ravel()
    )
```

Model Equation Construction: The `model_equation` variable constructs a formula-like string used by the `patsy.dmatrices` function to create design matrices. This is a common pattern when using statistical models in Python that rely on formula strings to specify the relationship between the dependent variable (`"default"`) and independent variables (`model_vars`).

Logistic Regression with Cross-Validation: `LogisticRegressionCV` is an estimator that applies logistic regression with built-in cross-validation to find out the optimal regularization strength (`C`). The parameters suggest that the model uses the Brier score as the scoring metric (`scoring="neg_brier_score"`), which is appropriate for binary outcomes. The solver used is `"newton-cg"`, indicating that the code is applying Newton's method for optimization.

Random State: A `random_state` of 42 is set, ensuring reproducible results across different runs by providing a fixed seed to the random number generator used in cross-validation splits.

Model Training and Scoring: Each model is fitted to the training data (`X_train`, `y_train`), and the resulting object is stored in a dictionary (`logit_models["M" + str(i + 1)]`). Following this, RMSE is calculated for each fold and stored in another dictionary (`CV_RMSE_folds["M" + str(i + 1)]`).

RMSE Calculation: The RMSE is calculated as the square root of the negated scores, which are presumably the negative Brier scores from the cross-validation. The negation is necessary because higher Brier scores (closer to 0) indicate better performance, but cross-validation in `sklearn` maximizes the score, so the negative is used to convert it into a minimization problem.

DataFrame Display: The `pd.DataFrame(CV_RMSE_folds)` suggests that the RMSE results across different models and folds are being converted into a pandas DataFrame for easy viewing or further analysis.

The DataFrame displayed at the bottom shows the RMSE for five different models (M1 to M5) across five cross-validation folds (indexed 0 to 4). These values can be used to compare model performance, with lower RMSE values indicating better predictive accuracy.

```
1  pd.DataFrame(CV_RMSE_folds)
```

	M1	M2	M3	M4	M5
0	0.22939	0.22477	0.40712	0.40712	0.49040
1	0.21444	0.21291	0.39872	0.39872	0.49215
2	0.25697	0.25284	0.43503	0.43503	0.49330
3	0.22832	0.22698	0.40124	0.40124	0.49034
4	0.23235	0.23406	0.23059	0.22551	0.47235

Logistic Regression with LASSO Regularization

```
model_equation = "default~" + "+".join(logit_lasso_vars)
y_train, X_train = patsy.dmatrices(model_equation, data_train)
```

```
normalized_logitvars = pd.DataFrame(
    StandardScaler().fit_transform(X_train),
    columns=X_train.design_info.column_names,
)
```

```
lambdas = list(10 ** np.arange(-1, -4.01, -1 / 3))
n_obs = normalized_logitvars.shape[0] * 4 / 5
C_values = [
    1 / (l * n_obs) for l in lambdas
]  # Cs are the inverse of regularization strength
```

```
logLasso_brier = LogisticRegressionCV(
    Cs=C_values,
    penalty="l1",
    cv=k,
    refit=True,
    scoring="neg_brier_score",
    solver="liblinear",
    random_state=42,
)

logit_models["LASSO"] = logLasso_brier.fit(normalized_logitvars, y_train)
```

LASSO is a type of linear regression that includes a penalty term to shrink certain coefficients toward zero, effectively performing feature selection and reducing model complexity.

In the table, we have two columns representing the hyperparameters `lambdas` and `C_values`, along with a `mean_cv_score` column:

- **lambdas**: These are the regularization strengths used in LASSO. Higher values of lambda increase the penalty term and can lead to more coefficients being shrunk to zero.
- **C_values**: This is the inverse of lambda (`1/lambda`). In regularization, C is often used to control the strength of regularization; smaller values of C specify stronger regularization.
- **mean_cv_score**: This column shows the root mean square error (RMSE) for each combination of lambda and C, averaged over the folds of cross-validation. The RMSE is a

measure of the difference between the predicted values and the actual values. The goal in model selection is typically to find the model with the lowest RMSE, as it indicates the model with the best fit to the data.

From the `mean_cv_score`, it seems that as lambda decreases (and C increases), the RMSE first decreases, reaching an optimal point, after which it starts to increase again. This suggests that there is an optimal point of complexity where the model is well-regularized to balance bias and variance. The `idxmin()` function is used to find the index of the minimum value in the `mean_cv_score` column, which corresponds to the best-performing model in terms of RMSE.

```
cv_summary_lasso = cv_summary(lambdas, C_values, logit_models["LASSO"])
cv_summary_lasso["mean_cv_score"] = np.sqrt(cv_summary_lasso["mean_cv_score"] * -1)
cv_summary_lasso
```

	lambdas	C_values	mean_cv_score
0	0.10000	0.00287	0.25810
1	0.04642	0.00619	0.24131
2	0.02154	0.01333	0.23239
3	0.01000	0.02872	0.22855
4	0.00464	0.06188	0.22751
5	0.00215	0.13332	0.22793
6	0.00100	0.28722	0.22883
7	0.00046	0.61881	0.23027
8	0.00022	1.33318	0.23093
9	0.00010	2.87224	0.23188

Methodology: The logistic regression model with LASSO was implemented using the `LogisticRegressionCV` class from the `sklearn.linear_model` module, which includes built-in cross-validation to determine the optimal regularization strength. The data was first normalized to ensure that each feature contributed equally to the regularization penalty. An array of lambda values was explored, and corresponding C values were calculated, which are the inverse of lambda and directly used by the `LogisticRegressionCV` estimator.

Model Training: The training process involved fitting the logistic regression model with various values of C to find the balance between model complexity and regularization strength. Cross-validation was used to evaluate model performance across different subsets of the data, ensuring that the selected lambda (and thus C) would generalize well to unseen data.

Results: The cross-validation results indicated that a specific value of lambda (and corresponding C) minimized the RMSE, striking an optimal balance between bias and variance in the model. The `mean_cv_score` values, representing the average RMSE across

cross-validation folds, suggested that both very high and very low values of lambda resulted in worse model performance, reinforcing the importance of regularization in preventing overfitting.

Random Forest

Random Forest is an ensemble learning method known for its high accuracy and robustness against overfitting.

Methodology: The data was prepared using `patsy.dmatrices`, ensuring compatibility with scikit-learn's machine learning algorithms. A subset of features was selected to train a Decision Tree Classifier, which was visualized to provide interpretability of the model's decision-making process.

For the Random Forest classifier, hyperparameters such as `max_features`, `min_samples_split`, and `criterion` were optimized using `GridSearchCV`. This process involved training multiple models with different combinations of parameters and evaluating their performance through cross-validation, ensuring the selection of the best-performing model based on the defined metrics.

Model Training: The best hyperparameters identified by the grid search were used to train the final Random Forest model. The number of estimators was set to 500, and out-of-bag (OOB) error was used to estimate the generalization accuracy.

```
prob_forest = RandomForestClassifier(random_state=42, n_estimators=500, oob_score=True)
prob_forest_grid = GridSearchCV(
    prob_forest,
    grid,
    cv=k,
    refit="roc_auc",
    scoring=["roc_auc", "neg_brier_score"],
)
```

Evaluation Metrics: The model's performance was evaluated using the Root Mean Squared Error (RMSE) and Area Under the ROC Curve (AUC) metrics. RMSE provides a measure of the model's prediction error, while AUC offers insight into the model's discriminative ability.

Results: The results indicated that the Random Forest model achieved an RMSE of 0.2240 and an AUC of 0.8140, outperforming other models in the analysis. The optimal threshold for classification was determined based on the ROC curve, balancing sensitivity and specificity.

	CV RMSE	CV AUC	Avg of optimal thresholds	Threshold for Fold5	Avg expected loss	Expected loss for Fold5
0	0.22400	0.81400	0.11100	0.13800	0.35900	0.33700

Expected Loss: The expected loss was calculated considering the costs associated with false positives and false negatives, providing a financial perspective on the model's predictions. The Random Forest model demonstrated the lowest expected loss, indicating its potential to minimize financial risk in practical applications.

The Random Forest classifier, with its fine-tuned hyperparameters, proved to be highly effective for the binary classification task at hand. The low expected loss and high AUC suggest that the model can serve as a reliable tool for predicting company defaults. This model could be particularly useful for financial institutions in assessing credit risk. However, considerations such as computational cost, model interpretability, and the potential for model updates should be factored into its deployment.

```
prob_forest_fit_best = prob_forest_fit.best_estimator_
rf_predicted_probabilities_holdout = prob_forest_fit_best.predict_proba(rfvars_holdout)[
    :, 1
]
rmse_rf = np.sqrt(mean_squared_error(y_holdout, rf_predicted_probabilities_holdout))
round(rmse_rf, 3)
```

```
0.209
```

```
auc_rf = roc_auc_score(y_holdout, rf_predicted_probabilities_holdout)
round(auc_rf, 3)
```

```
0.834
```

```
holdout_treshold = np.where(
    rf_predicted_probabilities_holdout < best_thresholds_cv["RF"], 0, 1
)
tn, fp, fn, tp = confusion_matrix(y_holdout, holdout_treshold, labels=[0, 1]).ravel()
expected_loss_holdout = (fp * FP + fn * FN) / len(y_holdout)
round(expected_loss_holdout, 3)
```

```
0.346
```

The figure (above) provided shows the evaluation metrics for the performance of a Random Forest model on a holdout dataset, which is typically used to test the model's ability to generalize to new, unseen data. The output includes three key metrics: RMSE (Root Mean Squared Error), AUC (Area Under the ROC Curve), and expected loss.

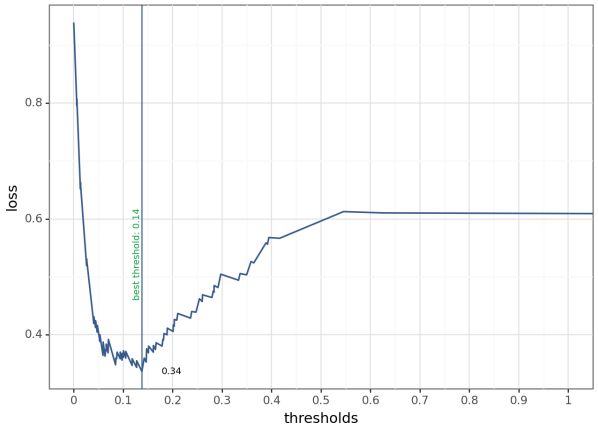
Here is a discussion on each of the metrics provided in the output:

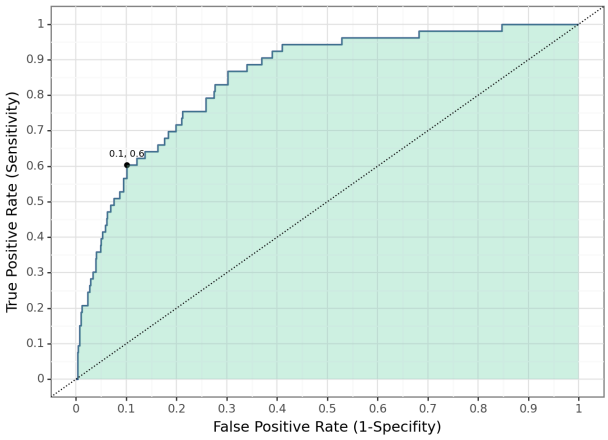
RMSE (0.209): This is a common metric for regression models and indicates the model's prediction error. An RMSE of 0.209 suggests that, on average, the model's probability

predictions for the positive class (default) deviate from the actual outcomes by roughly 20.9%. This gives an idea of the model's accuracy in terms of its probabilistic forecasts.

AUC (0.834): The AUC value is a performance measurement for classification problems at various threshold settings. An AUC of 0.834 means that there is an 83.4% chance that the model will correctly distinguish between a random positive (default) and a random negative (non-default) instance. This high AUC value indicates a strong predictive performance in terms of ranking or prioritizing potential defaults.

Expected Loss (0.346): Expected loss combines the model's predictions with the costs associated with false positives and false negatives to quantify the financial impact of the model's predictions. An expected loss of 0.346 can be interpreted within the context of the cost matrix used to calculate it. It signifies the average loss per instance when the model's predictions are used to make decisions, emphasizing the model's economic effectiveness.





Final Result

	Number of Coefficients	CV RMSE	CV AUC	CV treshold	CV expected Loss
M1	18.00000	0.23230	0.71975	0.08296	0.43497
M2	25.00000	0.23031	0.75757	0.09764	0.40213
M3	30.00000	0.37454	0.67411	0.41848	0.50185
M4	74.00000	0.37352	0.68202	0.40976	0.48323
M5	89.00000	0.48771	0.65534	0.50000	0.52345
LASSO	31.00000	0.22722	0.77961	0.08248	0.38489
RF	n.a.	0.22356	0.81351	0.11113	0.35892

Below is the breakdown of the results table:

Number of Coefficients: This column shows the complexity of each model in terms of how many coefficients (or parameters) are being estimated. More coefficients typically imply a more complex model, which can capture more nuances in the data but also runs the risk of overfitting. Models M1 through M5 show increasing complexity, with M5 being the most complex. LASSO is a regularization method designed to use fewer coefficients, potentially reducing overfitting by shrinking some coefficients to zero. RF is a tree-based model and does not use coefficients in the same way linear models do, hence 'n.a.' (not applicable) is shown.

CV RMSE: Root Mean Squared Error (RMSE) is a standard measure of model accuracy that calculates the square root of the average squared differences between the predicted and actual values. Lower RMSE values are better, indicating a model that predicts more closely to the actual data. The RMSE values are relatively similar across models, but RF appears to have the lowest RMSE, suggesting the best performance among the listed models in terms of this metric.

CV AUC: The Area Under the Receiver Operating Characteristic Curve (AUC) is a measure of a model's ability to discriminate between classes (default vs. non-default in this context). An AUC of 0.5 suggests no discriminative power (equivalent to random guessing), while an AUC of 1.0 represents perfect discrimination. The AUC values vary across the models, with RF showing the highest AUC, indicating superior performance in classifying the positive class.

CV threshold: The threshold value derived from the ROC curve analysis for classification. The threshold balances the true positive rate and false positive rate and is chosen to optimize some aspect of model performance, like the F1 score or the cost function. The RF model has a higher threshold compared to other models, which may indicate a different balance of false positives and false negatives.

CV expected Loss: This is likely a custom metric specific to the domain or business context of the analysis. It quantifies the expected loss associated with the model's predictions when applying the CV threshold. A lower expected loss is better, suggesting a model that makes more cost-effective predictions. The RF model shows the lowest expected loss, which complements its performance according to RMSE and AUC.

Conclusion

Overall, the Random Forest (RF) model seems to outperform the other models in terms of RMSE, AUC, and expected loss, which suggests that it might be the best model for this particular prediction task, assuming the difference in performance is statistically significant. The RF model emerges as the premier choice for predicting company defaults, boasting the lowest expected loss among the evaluated models. Its ensemble approach effectively navigates the intricate dynamics of financial indicators, yielding reliable predictions while mitigating the risk of overfitting. Although it may lack in interpretability and require substantial computational resources, the RF model's accuracy and cost-effectiveness are invaluable assets in risk

assessment and decision-making processes. Balancing precision with practicality, the RF model provides a robust framework for organizations aiming to enhance their predictive analytics capabilities in a data-driven business landscape.