BANA 620 PROJECT REPORT: DESCRIPTIVE AND PREDICTIVE ANALYTICS APPLIED TO THE SKILLED NURSING FACILITY COST REPORTS

Joshua Cabal

California State University, Northridge

April 16, 2024

Contents

1 Executive Summary

An overview of the project, including key findings, recommendations, and a brief summary of the analysis conducted. This section should be concise and geared toward readers who may not delve into the full details of the report.

2 Introduction

Context and background information about the opportunity the project addresses. This section should outline the objectives of the project and the significance of the analysis being conducted.

2.1 Context and Background Information

2.2 Objectives

3 Methodology

A detailed description of the analytical methods and tools used in the project. This should include data collection processes, data analysis techniques (e.g., statistical methods, machine learning algorithms), and any software or programming languages utilized.

3.1 Software

The entire analysis was conducted using Python (3.11.4) within the Jupyter Notebooks environment. The following Python libraries were used in the project:

1. **Pandas** for data manipulation and cleaning.
2. **NumPy** for numerical operations.
3. **Scikit-learn** for implementing machine learning algorithms and preprocessing methods.
4. **Matplotlib** and **Seaborn** for data visualization, facilitating the interpretation of results and comparison of model performance.

These tools were selected for their robust functionality and widespread adoption in the data science community, enabling efficient data processing and modeling capabilities.

3.2 Preprocessing Scalers

Two different numerical scalers were used and compared. Below is the description of each method.

1. **Standard Scaler:** Standardize features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:

$$z = \frac{x - u}{s}$$

   Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform.

   Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).

   For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

2. **Robust Scaler:** Scales features using statistics that are robust to outliers.

   This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).

   Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Median and interquartile range are then stored to be used on later data using the transform method.

   Standardization of a dataset is a common preprocessing for many machine learning estimators. Typically this is done by removing the mean and scaling to unit variance. However, outliers can often influence the sample mean / variance in a negative way. In such cases, using the median and the interquartile range often give better results.

3.3 Machine Learning Models

Severl different models were used and compared in this study to determine model suitability. Below are the details related to each method.

1. **Linear Regression**: Ordinary least squares Linear Regression. LinearRegression fits a linear model with coefficients w = (w1, ..., wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

$$\text{minimizes: } \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})\right)^2$$

   where:

   - n is the number of observations in the dataset.
   - y_i represents the observed value of the dependent variable for the i-th observation.
   - x_{ij} is the value of the j-th explanatory variable for the i-th observation.
   - β_j denotes the coefficient for the j-th explanatory variable, quantifying the effect of this variable on the response.
   - β_0 is the intercept, representing the expected mean value of y_i when all explanatory variables are equal to zero.

2. **Lasso Regression**: Linear Model trained with L1 prior as regularizer (aka the Lasso). Regularization improves the conditioning of the problem and reduces the variance of the estimates. Larger values specify stronger regularization.

$$\text{minimizes: } \left\{ \frac{1}{2n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \alpha\|\mathbf{w}\|_1 \right\}$$

   where:

   - n represents the number of observations in the dataset.
   - y_i is the observed value of the dependent variable for the i-th observation.
   - $\mathbf{x}_i$ is the vector of explanatory variables (features) for the i-th observation.
   - $\mathbf{w}$ denotes the vector of coefficients (including the intercept and slopes) of the linear model.
   - α is a non-negative regularization parameter that controls the strength of the L1 penalty, encouraging sparsity in the vector of coefficients.
   - $\|\mathbf{w}\|_1$ represents the L1 norm of the coefficients, which is the sum of the absolute values of the coefficients.

3. **Ridge Regression**: Linear least squares with L2 regularization. This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm

$$\text{minimizes: } \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

   where:

- n is the number of observations in the dataset.
- y_i represents the observed value of the dependent variable for the i-th observation.
- x_{ij} is the value of the j-th explanatory variable for the i-th observation.
- β_j denotes the coefficient for the j-th explanatory variable, quantifying the effect of this variable on the response.
- β_0 is the intercept, representing the expected mean value of y_i when all explanatory variables are equal to zero.
- λ is a non-negative regularization parameter that controls the strength of the penalty applied to the size of the coefficients.

4. **Elastic Net**: Linear regression with combined L1 and L2 priors as regularizer.

$$\text{minimizes: } \frac{1}{2n}\sum_{i=1}^{n}\left(y_i - (\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip})\right)^2 + \lambda\left(\alpha\sum_{j=1}^{p}|\beta_j| + \frac{1-\alpha}{2}\sum_{j=1}^{p}\beta_j^2\right)$$

where:

- n is the number of observations in the dataset.
- y_i represents the observed value of the dependent variable for the i-th observation.
- x_{ij} is the value of the j-th explanatory variable for the i-th observation.
- β_j denotes the coefficient for the j-th explanatory variable, quantifying the effect of this variable on the response.
- β_0 is the intercept, representing the expected mean value of y_i when all explanatory variables are equal to zero.
- λ is a non-negative regularization parameter that controls the overall strength of the penalty.
- α is a parameter between 0 and 1 that balances the contribution of L1 and L2 penalties: α for the L1 component and $(1-\alpha)$ for the L2 component.

5. **K Neighbors Regressor**: Regression based on k-nearest neighbors. The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

$$\text{prediction for } x_q = \frac{1}{K}\sum_{i \in N_K(x_q)} y_i$$

- x_q is the query point for which the prediction is to be made.
- K is the number of nearest neighbors considered for making the prediction.
- $N_K(x_q)$ is the set of indices of the K nearest neighbors to x_q.
- y_i are the observed values of the dependent variable for the neighbors.

4 Data Description

An overview of the data set(s) used, including sources, size, and characteristics of the data. Highlight any data cleaning or preprocessing steps undertaken to prepare the data for analysis.

4.1 Skilled Nusing Facility Cost Report

Medicare-certified institutional providers are required to submit annual cost reports. These data files contain the highest level of cost report status for cost reports in each reported fiscal years. The cost report contains provider information such as facility characteristics, utilization data, cost and charges by cost center (in total and for Medicare), Medicare settlement data, and financial statement data. CMS maintains the cost report data in the Healthcare Provider Cost Reporting Information System (HCRIS). Skilled Nursing Facilities (SNF) submit their cost report data to HCRIS using form CMS-2540-2010. The reports used are from the years 2015 through 2021.

4.1.1 Data Loading and Handling Missing Values

Each file comprises annual cost reports for over 15,000 skilled nursing facilities, all stored in separate CSV files for each of the years under study. To facilitate analysis, these files required concatenation into a single DataFrame. Initially, inconsistencies in column headers across the files for the years 2020 and 2021 were modified to match with those of previous years. This matching was manually executed in Microsoft Excel by renaming similar columns accordingly and discarding unmatched columns. Additionally, a `Year` attribute was appended to each record during the import step to preserve the, and all column headers were converted to lowercase to ensure consistency.

Following the import, the primary focus shifted to managing null values within the dataset. The initial step involved column-level cleaning, where I established two distinct lists of columns slated for removal: `dropNull` and `dropRedundancy`. The `dropNull` list encompassed columns that exhibited excessive nullity, with more than 90% of their values missing across all records. The `dropRedundancy` list comprised columns that provided redundant information, which was otherwise encapsulated by other attributes. Given that the financial forms required a detailed breakdown of reported figures, only the most aggregated data were retained by removing the redundant columns.

To address the remaining null values, record-level cleaning was conducted through simple imputation. Specifically, mean imputation was employed, and the imputed values were rounded to the nearest integer. This rounding was justified as several of the attributes inherently attain only integer values. After these preprocessing steps, the dataset was streamlined from an initial 100 columns to 55, thereby enhancing the manageability and potential predictive accuracy of the ensuing models.

4.1.2 Handling Outliers

The initial approach to managing outliers involved the application of a z-score threshold method. This technique operates by setting a specific z-score limit and excluding all records that exhibit values beyond this predefined range. For this analysis, a threshold of 3 was selected, focusing specifically on the variables 'net income' and 'accounts payable.' This decision was informed by extensive testing across various attributes and different threshold settings, which demonstrated that targeting these particular columns effectively addressed a significant portion of the outliers.

```
# dealing with a subset of outliers by z-score threshold
def removeOutliers(input_df, column_name, zScoreThreshold):
    mean = input_df[column_name].mean()
    std = input_df[column_name].std()
    z_scores = (input_df[column_name] - mean) / std

    return input_df[(z_scores > -zScoreThreshold) & (z_scores < zScoreThreshold)]

numerical_columns = ['net_income', 'accounts_payable']
for each in numerical_columns:
   df = removeOutliers(fullCostReportdf, each, 3)
```

Listing 1: Function to clean by z-score threshold

Upon completion of the initial outlier management, a subsequent manual review of the dataset was conducted to identify any additional outliers. This examination revealed that the attribute `number of beds` contained suspect outliers. Specifically, eleven nursing facilities reported possessing in excess of 21,000 beds. To address this anomaly, a variable named bedThreshold was defined and set at 21,000. All records exceeding this threshold were subsequently removed from the dataset to ensure the integrity and accuracy of the analysis.

4.1.3 Feature Engineering

This study employs feature engineering techniques to derive additional predictive insights from the dataset's financial metrics. Recognizing that interactions among variables can potentially encapsulate a greater degree of variance than the individual variables in isolation, we have constructed additional features which aim to capture information related to the financial health of the nursing homes. These newly defined features, which capitalize on the interactions between existing variables, are posited to possibly enhance the predictive capacity of the resultant model. Specifically, the following features were engineered and incorporated into the analysis:

1. **Profit Margins:**

- Gross Profit Margin: (`gross_revenue` - `total_costs`) / `gross_revenue`.
- Net Profit Margin: `net_income` / `gross_revenue`.

2. **Operational Efficiency:**
   - Total Operating Expense Ratio: `less_total_operating_expense` / `gross_revenue`.
3. **Liquidity Ratios:**
   - Current Ratio: `total_current_assets` / `total_current_liabilities`.
4. **Solvency Ratios:**
   - Debt-to-Equity Ratio: `total_liabilities` / `total_fund_balances`.
5. **Return on Investment (ROI):**
   - ROI: `net_income` / `total_assets`.
6. **Capacity Utilization:**
   - Bed Occupancy Rate: `total_days_total` / (`number_of_beds` $\times$ 365).
7. **Cost Management:**
   - Cost per Bed: `total_costs` / `number_of_beds`.
   - Salary Costs per Bed: `total_salaries_adjusted` / `number_of_beds`.

In subsequent analyses, not all of the aforementioned features demonstrated statistical significance. This observation is elaborated upon in the later sections of this study, where the impact of each variable on the model's performance is evaluated.

4.1.4 Feature Scaling and Encoding

The two different scalers that were used were `StandardScaler()` and `RobustSclaer()`. The regressors were Linear, Lasso, Ridge, Elastic Net, and K Neighbors. Below is the code for the model implmentation:

```
X = df.drop(['net_income'], axis=1)
y = df['net_income']

# Define numerical features for scaling
numericalFeatures = X.columns[X.dtypes != 'object']

# Preprocessor using ColumnTransformer to apply scaling only to numerical features
preprocessor = ColumnTransformer(
    transformers=[
        ('num', RobustScaler(), numericalFeatures)
    ])

models = {
    'Linear Regression': LinearRegression(),
    'Ridge': Ridge(alpha=20000),
    'Lasso': Lasso(alpha=20000),
    'ElasticNet': ElasticNet(alpha=1.0, l1_ratio=0.5),
    'KNN': KNeighborsRegressor(n_neighbors=2)
}

results = pd.DataFrame(index=models.keys(), columns=['Mean Squared Error', 'R-squared'])

# Iterate over models, create a full pipeline, and evaluate
for name, model in models.items():
    pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                               ('regressor', model)])
    # Split data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
    random_state=123)
    # Fit the model
    pipeline.fit(X_train, y_train)
    # Predict
```

```
    y_pred = pipeline.predict(X_test)
    # Evaluate
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results.loc[name] = [mse, r2]

print(results)

# Plotting results
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

results.plot(kind='bar', y='Mean Squared Error', ax=ax[0], color='skyblue', title='Mean Squared Error by Model')
results.plot(kind='bar', y='R-squared', ax=ax[1], color='orange', title='R-squared by Model')

plt.tight_layout()
plt.show()
```

Listing 2: Regression model pipeline code

5 Analysis and Findings

Presentation of the analysis conducted, including data visualization (charts, graphs, tables) and statistical outputs. This section should detail the insights gained from the analysis, interpreting the results in the context of the business problem.

6 Discussion

Interpretation of the findings, discussing how they address the project objectives and their implications for the business. This section should also cover any limitations of the analysis and considerations for future research.

7 Recommendations

Based on the analysis and findings, provide actionable recommendations for the business. Clearly articulate the expected impact of these recommendations and suggest a plan for implementation.

8 Conclusion

Summarize the key points from the report, reinforcing the value of the findings and recommendations.

9 Appendices

Include any additional material that supports the analysis, such as detailed data tables, code snippets, or extended methodology descriptions.

10 References

List all sources cited in the report, including data sources, literature, and any external references used in the analysis.

```python
# Loop through each year, read the CSV file, add a 'Year' column, and append to the list
def get_file_path():
    if os.name == 'nt':  # Windows
        return r'C:\Users\joshu\OneDrive\Desktop Files\Textbooks and Syllabi\CSUN Semester 6\BANA 620\Project\Data'
    else:  # macOS or other Unix-like OS
        return '/Users/josh/Library/CloudStorage/OneDrive-Personal/Desktop Files/Textbooks and Syllabi/CSUN Semester 6/BANA 620/Project/Data'

base_path = get_file_path()
years = ['2015', '2016', '2017', '2018', '2019']
dataframes = []

for year in years:
    file_path = os.path.join(base_path, f'{year}_CostReport.csv')
    df = pd.read_csv(file_path)
    df['Year'] = year  # Add a 'Year' column
    dataframes.append(df)

# Concatenate all DataFrames into one
costReportdf = pd.concat(dataframes, ignore_index=True)
```

Python

Figure 1: Data loading code snippet