Preprocessing/Feature Engineering Revised

Summary: Basic and more advanced feature engineering are performed on APR dataset. Imputation was performed on several features with missing values. This is a rewrite of a previous version. Soft impute is a matrix completion algorithm for missing values based off soft thresholding. Previously, imputations were performed one col at a time - totally unnecessary. Additionally, the normalization using StandardScalar() (required ahead of imputation) preserves nulls. Therefore, the custom script was also unnecessary.

This version abstracts away most of the engineering/cleaning into classes that are (hopefully) much easier to follow and continue to evolve. Additionally, wrappers are placed around SoftImpute and StandardScalar to allow sklearn pipeline use.

Student: Jake Anderson

Jupyter Notebook: `Imputation_Engineering.ipynb` and `SVD_ex.ipynb`

Result:

1. Imputation on several features
2. Engineered: Frequency count, binned, and reduced cardinality categorical features

Notation

The dataset is presented in various stages of preprocessing and is generally represented as $\mathbf{X} \in \mathbb{R}^{N \times D}$. Additionally, the dataset is often subsetted for different phases: training, testing, and validating. Subsets are represented as a set $(\mathbf{X}^{(\text{phase})}, \mathbf{y}^{(\text{phase})})$ such that $\mathbf{X}$ is a matrix of features and $\mathbf{y}$ is the response vector. Models $\mathcal{P}$ treats samples as vectors of features which can be represented as sets $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}\}$. Feature vectors are then vectors of samples represented as $\{\mathbf{x}_1, ..., \mathbf{x}_D\}$ and can also be expressed with matrix notation $\mathbf{X}_{:,j}$. Therefore, a feature value j for sample i is represented $\mathrm{x}_j^{(i)}$.

Jupyter Notebook Walkthrough: `Imputation_Engineering.ipynb`

The following steps correspond to the cell blocks found in `Imputation_Engineering.ipynb`

1. Import data, subset dataframe, separate label, identity non-course features for inclusion

2. define dataframes: CourseSeparater( ) class used, FeatureMaps( ).

3. Pipeline walkthrough: Dissects the pipelines a bit before using them. SoftImpute( ) is included in the numerical pipe. With the pandas wrapper, it is fairly user friendly.

4. Preprocessing Pipeline: All pipes are listed. QuickPipeline_mod( ) is modified from its github version to implement the fit_transform( ) method. Additionally, it is included separately in certain pipes due to an issue with hot encoding bools. I'll need to dive in to fix it eventually.

5. Final output $\mathbf{X} \in \mathbb{R}^{\approx 3800 \times 162}$

Code Folder

`APR_project_final/code_datalab`

The `code_datalab` folder contains 4 sub-folders. The first cell in every notebook imports all libraries and scripts in this folder. However, this folder should be explored, understood, and ultimately improved upon.

1. `features.py` Contains some random scripts. The `make_dict` and `freq_group` are useful and used in feature engineering. The most important script in this file is the scripts related to learning curves. If problems arise down the road, this script can be found online (github).

2. `datalab.py` Contains scripts specific to APR. As of now, this automatically imports. However, this file should be explored, updated thoroughly.

3. `quickpipeline.py` Pulled from github originally. The code is simple enough and saves a ton of preprocessing time. Be sure to account for data leakage when fitting. If in doubt, remove the response entirely prior to using this script. As mentioned, this has been modified to implement the fit_transform( ) method.

4. `pandas_feature_union.py`: Slightly modified for use in sklearn pipelines

5. `libraries.py` Straight forward. Adjust as needed. Be sure to understand `sys.path.append`.

1. Dataset Information

Dataset description: $\mathbf{X} \in \mathbb{R}^{3800\times160}$ mixed datatypes and $\mathbf{y}$ is the binary response vector = whether the student met the APR defined threshold $\mathbf{x}^{(i)} = 1$. The APR dataset is a 2:1 imbalanced[1] subset of STEM students s.t. cohort 2009 - 2017.

2. Features

Table 1: *Main Features*

Cohorts: 09 - 17		
APR	`apr`	binary, response
Student ID	`stu_id`	categorical, anonymized
Student Age	`stu_age`	numerical
Classes	`class_taken`	binary, ≈ 200
GPA/Tests	`best_test`	numeric, best of SAT or ACT
	`hs_gpa`	ordinal, high school gpa
	`und_gpa`	ordinal, cum for summer/fall
	`fall_gpa`	ordinal, Fall gpa
Financial	`fin_`	numeric, imputation needed

2.1 Engineering

Initial preprocessing involved screening samples for usability, removing null values, identifying missing values, and (in some cases) removing extreme or error values. Duplicate values were removed and features with missing values were assessed on a case-by-base basis. The term *data leakage* is not formally defined and is used in a variety of contexts. For APR analysis, leakage was defined as the unintentional injection of response

[1] 2655 non-churn to 1212 churn

data into a training set or predictor. The inclusion of features that unintentionally have response-related patterns often leads to misleading performance metrics that will ultimately reflect in poor generalization.

Basic feature engineering methods used on the APR dataset included: equal-frequency binning, feature value aggregation, and feature transformation.

The concept of equal-frequency binning groups continuous features by intervals and assigns a frequency count, therefore extracting additional distribution related information [34]. Frequency counts can also serve as a means of combining categories values for computational efficiency; similar to *stemming* for text classification shown by [17]. [34] highlights the computational efficiency of equal-frequency binning for tree-based models given their use of order statistics (for comparison at decision points). Less common categorical values can be identified by their frequency count and merged to increase data tractability, especially in the instance of high-cardinality[2] categorical features. Kang et al. [33] introduced several data aggregation techniques involving merging homogenous values of different predictors. The consolidation of categorical data based off relative frequency was performed on county, state, banner, and department data (cell 11). For the purposes of this analysis, aggregation methods for low relative frequency values for higher cardinality features are used on the APR dataset; resulting in the use of categorical features that would otherwise cause overfitting.

Equal-frequency binning [34] is typically used to convert numerical features into categorical features. A continuous interval is divided into K intervals with approximately equal occurrence frequency counts. The interval is then defined as a bin and assigned to the continuous feature as a category. This is not necessarily a replacement of the continuous feature, but rather a method to obtain more spatial information in the form of frequency distributed categories. Equal frequency binning was applied to `stu_age`, `stu_load`, `AP_cred` and all financial/gpa (cells 9-11).

[2]Categorical feature with feature values > 100

2.2 Imputation

The `Soft Impute` algorithm uses nuclear norm $||\cdot||_*$ regularization to approximate and fit a low rank matrix to a matrix with missing values [45]. The algorithm uses approximations to fill in missing values while minimizing the rank of the matrix. For matrix completion, the nuclear norm behaves similar to a ℓ_1 penalty by shrinking singular values. This allows for more dimensions without the cost of higher variance. Using the `Soft Impute` algorithm, the missing values for financial/gpa missing values were imputed (see Appendix: Table 2). The policy dataset was processed for imputation to avoid data leakage and confounding variables by removing the response and categorical features. The feature for imputation became the response feature and all other numerical values were normalized using a preprocessing pipeline [53]. The python translated `softImpute` version of the original R package authored by Hastie et al. [26] was obtained through github.

Algorithm 1 Soft-Impute

Require: Grid of warm starts λ **s.t.** $\lambda_1 > \lambda_2 > ..... > \lambda_k$

function SOFT IMPUTE(X)
    solutions $\leftarrow \{\}$
    Initialize Z with a matrix of zeros, $Z_{old} \leftarrow 0$
    for i from 1 to k (for λ values) **do**
        $\hat{X} \leftarrow P_\Omega(X) + P_\Omega^\perp(Z_{old})$
        $Z^{new} \leftarrow U\mathbf{S}_\lambda(D)V^T \leftarrow \mathbf{S}_\lambda(\hat{X})$
        Define the error threshold
        while $\frac{||Z^{new} - Z^{old}||_F^2}{||Z^{old}||_F^2} > \tau$ **do**
            $Z_{old} \leftarrow Z_{new}$
            $\hat{X} \leftarrow P_\Omega(X) + P_\Omega^\perp(Z_{old})$
            $Z_{new} \leftarrow U\mathbf{S}_\lambda(D)V^T \leftarrow \mathbf{S}_\lambda(\hat{X})$
        The matrix Z with imputed missing values attains the desired error
        $\hat{Z}_{\lambda_i} \leftarrow Z_{new}$
        solutions.append($\hat{Z}_{\lambda_i}$)
    return solutions

2.4 Notes

Soft-thresholding is a leading imputation technique and is highly recommended. It is often times avoided in favor of easier more explainable (but less effective) methods. See appendix for additional notes. Additionally, `SVD_ex.ipynb` is included as a simplified version of soft-thresholding.

APPENDIX

A.1

Let $\mathbf{X} \in \mathbb{R}^{N \times D}$ be a numerical matrix with missing values, rank $= r$, singular value decomposition (SVD) $= UDV'$, and the set of indices of observed entries $\Omega = \{(i,j) : \mathbf{X}_{ij} \text{ observed}\}$. [27] defined a projection matrix $P_\Omega(\mathbf{X})$ with the observed elements of $\mathbf{X}$ preserved and missing values = 0. Conversely, $P_\Omega^\perp(\mathbf{X})$ was the defined projection matrix onto the complement of the set Ω. Therefore, observed elements from $\mathbf{X}$ are set at zero and placeholders (later to be approximated) values replace missing values. Since the placeholders are essentially the same as missing values, the following holds: $\mathbf{X} = P_\Omega^\perp(\mathbf{X}) + P_\Omega(\mathbf{X})$. We can then define the SVD *soft-thresholding* function $\mathbf{S}_\lambda(\mathbf{X})$ as:

(i) Given SVD **s.t.** $\mathbf{X} = UDV^T$

(ii) $\mathbf{S}_\lambda(\mathbf{X}) \equiv U\,\mathbf{S}_\lambda(D)V^T$ **s.t.** $\mathbf{S}_\lambda(D) = diag[(d^{(1)} - \lambda)_{[+]}, ..., (d^{(r)} - \lambda)_{[+]}]$

We can then begin updating the matrix $\hat{\mathbf{X}}$ with approximated missing values using an iteratively updating matrix $\hat{\mathbf{Z}}$; which can be initialized as a zero matrix:

(i) $\hat{\mathbf{X}} \leftarrow P_\Omega(\mathbf{X}) + P_\Omega^\perp(\hat{\mathbf{Z}})$

(ii) $\hat{\mathbf{X}} = UDV^T$

(iii) $\hat{\mathbf{Z}} \leftarrow U\mathbf{S}_\lambda(D)V^T$

[45] defined the following convex optimization function:

$$f_\lambda(Z) := \underset{Z}{\text{minimize}} \frac{1}{2}||P_\Omega(X) - P_\Omega(Z)||_F^2 + \lambda||Z||_* \tag{1}$$

Equation (1) can be solved algorithmically using Algorithm 1.

Table 2: *Features: Imputation*

GPA	`hs_gpa`	ordinal, high school gpa; 900 nan
	`fall_gpa`	ordinal, Fall gpa; 1315 nan
Financial	`fin_inc`	numeric, financial income; 512 nan
	`fin_aid`	numeric, financial aid needed; 484 nan
	`fin_awd`	numeric, financial award; 484 nan