UNDERSTANDING COMPUTER SCIENCE ACADEMIC PERFORMANCE USING PRINCIPAL COMPONENTS ANALYSIS

A Thesis
by
CHRISTOPHER SMITH

Submitted to the Graduate School
Appalachian State University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

December 2017
Department of Computer Science

UNDERSTANDING COMPUTER SCIENCE ACADEMIC PERFORMANCE USING PRINCIPAL COMPONENTS ANALYSIS

A Thesis
by
CHRISTOPHER SMITH
December 2017

APPROVED BY:

_____
R. Mitchell Parry, Ph.D.
Chairperson, Thesis Committee

_____
Alice McRae, Ph.D.
Member, Thesis Committee

_____
Rahman Tashakkori, Ph.D.
Member, Thesis Committee

_____
Rahman Tashakkori, Ph.D.
Chairperson, Department of Computer Science

_____
Max C. Poole, Ph.D.
Dean, Cratis D. Williams School of Graduate Studies

Abstract

UNDERSTANDING COMPUTER SCIENCE ACADEMIC PERFORMANCE USING PRINCIPAL COMPONENTS ANALYSIS

Christopher Smith
B.S., Appalachian State University
M.S., Appalachian State University

Chairperson: R. Mitchell Parry, Ph.D.

Some students perform better in school than others. Some classes are also harder than others. This thesis poses the question: Are there types of students that do better in certain types of classes? We model student grades as a combination of class difficulty, student GPA, and student-class preference using student transcript data for Computer Science undergraduates at Appalachian State University. This thesis applies principal components analysis to relate classes to each other, interprets these relationships, and quantifies their importance for grade estimation.

Acknowledgements

My special thanks go to Dr. Parry for his insight on recommendation systems and machine learning, as well as the numerous hours revising this thesis. I would also like to thank Dr. Tashakkori and Dr. McRae for the time and effort they took to read and help develop the content of this thesis. The Department of Computer Science at Appalachian State University, the Lowe's Distinguished Professor Research Fund, and the NSF S-STEM program have provided me financial support, making my higher education possible. Thanks should also go to all of the authors of the cited publications. Lastly, I would like to sincerely thank my family and friends for their support.

Contents

Chapter 1 - Introduction

The six-year graduation rate for undergraduate students in public universities was 57% in 2011 [1]. Changing major, failing a class, and overscheduling can affect timely graduation. With a better understanding of the types of students and types of classes, students could make better, more informed decisions and potentially increase graduation rates. For example, grade prediction systems provide one way to utilize relationships between students and classes.

Grade prediction systems can use patterns in historical transcript data to estimate future grades. For example, Chamillard used linear regression to predict computer science course grades based on other classes students had taken [2]. This required training on a set of students who had all taken the same classes. In a more general scenario, two students are unlikely to take all the same classes when a university offers thousands of classes, but each student only takes about 40. This problem is closely related to those solved by recommendation systems. For example, Netflix's recommendation system [3] estimates the ratings for thousands of movies per user, where each user only rates tens or hundreds. One approach uses matrix factorization [4] to aggregate ratings of similar items and similar users. This thesis applies the same approach to estimate student grades.

Matrix factorization has been used to estimate student grades [5, 6, 7]. These approaches project students and classes into a low-dimensional vector-space. These di-

mensions can be understood as characteristics of classes or students that inform grade estimation. Although the prior work demonstrated the usefulness of the approach, it did not attempt to visualize or interpret these dimensions to substantiate their use. This thesis attempts to fill this gap in the research using historical grades in computer science classes at Appalachian State University.

This thesis compares several models to estimate student grades. First, we consider a model that uses the university mean only, which serves as a baseline comparison for the other models. Second, we consider a model that uses the average grade for each course. A third model uses the average grade for each student. The fourth model combines student averages and course averages to estimate grades. Our fifth model uses matrix factorization to infer interactions between classes and the students who perform well in them. Finally, we compare each model's performance on two different data sets.

The following chapters provide methods, results, and conclusions. Chapter 2 provides details for each model. Chapter 3 compares the models on the data sets, while Chapter 4 interprets the results and makes suggestions for future work.

Chapter 2 - Methodology

This chapter describes the data used, the models considered, how their parameters are estimated, and the methods for comparison.

2.1 The Data Sets

This thesis obtained the academic transcripts for 16,000 students who have taken at least one computer science class from the institutional research office at Appalachian State University. We parsed these data into a matrix, where each row corresponds to a student and each column corresponds to a course. We only considered a student's first attempt at each class. We then created two subsets of these data for use in this thesis. The first data set contained only classes that appear among computer science major requirements. These include mathematics, science, and computer science courses. Then, we ignored classes that had fewer than 50 total students and students who had taken fewer than 10 classes. This data set contained 1,177 students and 38 classes. The second data set contained the 38 classes from the first data set and an additional 32 classes with the largest enrollment. After filtering students with fewer than 10 classes, this set contained 9,554 students and 70 classes.

2.2 The Models

The $M \times N$ matrix $\mathbf{X}$ contains the grades such that x_i^j is the grade received by student i in class j, where M is the number of students and N is the number of classes:

$$\mathbf{X} = \begin{bmatrix} x_1^1 & x_1^2 & \dots & x_1^N \\ x_2^1 & x_2^2 & \dots & x_2^N \\ \vdots & \vdots & \ddots & \vdots \\ x_M^1 & x_M^2 & \dots & x_M^N \end{bmatrix} \tag{2.1}$$

The following figure shows a heatmap of $\mathbf{X}^T$ sorted by student GPA (left-to-right) and class average (top-to-bottom). Colors range from yellow to red, representing grades from F to A. The black areas indicate that a student has not taken a class, i.e., missing values.

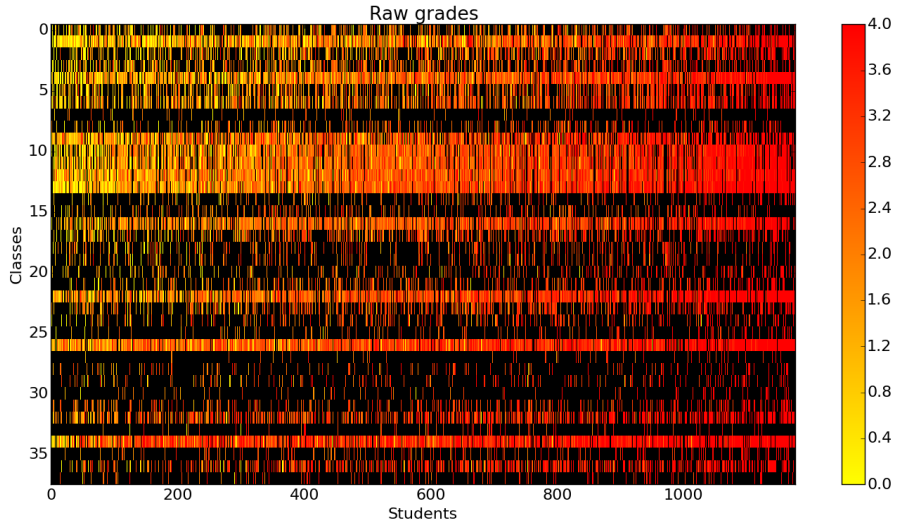

Figure 2.1: Heatmap of grades for the computer science data set

Each of our proposed models optimize the following general cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_i^j - \hat{x}_i^j \right)^2 \left\{ x_i^j \text{ is known} \right\} + \lambda R(\theta) \tag{2.2}$$

In this equation, θ represents the parameters for each model, and $\hat{x}_i^j$ represents the model's estimate for the grade of student i in class j. We select parameters to reduce the sum of the squared errors between the actual grade, x_i^j, and the estimated grade, $\hat{x}_i^j$. Because the vast majority of grades are treated as missing values, the summations only include those grades that are known, designated by the indicator function in curly braces. $R(\theta)$ is a regularization term which biases the optimization toward simpler models with parameter values near zero. The parameter λ controls how strongly parameter values are pulled toward zero and therefore complexity of the resulting model. This helps to reduce overfitting on small data sets [8].

2.2.1 The m Model

First we consider a simple model that estimates the same grade for everyone using the following criterion function:

$$J(m) = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_i^j - m \right)^2 \left\{ x_i^j \text{ is known} \right\}, \tag{2.3}$$

where $\hat{x}_i^j = m$. In this case, the value of m that minimizes the criterion is the mean of all known grades:

$$m = \frac{1}{G} \sum_{i=1}^{M} \sum_{j=1}^{N} x_i^j \left\{ x_i^j \text{ is known} \right\}, \tag{2.4}$$

where G is the total number of known grades. Figure 2.2 shows a heatmap of the model residuals. The blue elements of the matrix indicate underestimates, whereas red elements indicate overestimates. Colors near white indicate close estimates. Residuals decrease from the upper-left to the lower-right corner. That is, students and classes with high averages tend to get underestimated the most, whereas students and classes with lower averages are overestimated the most.

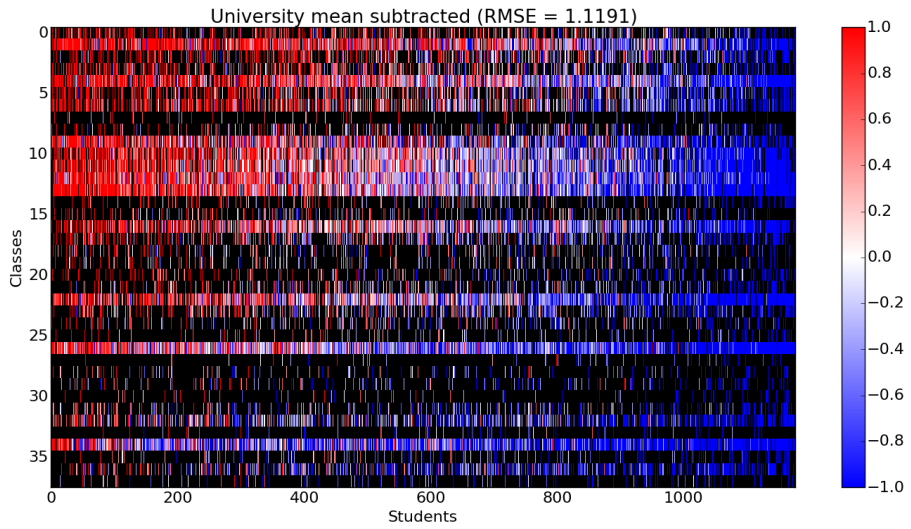

Figure 2.2: Heatmap of residuals for the m model

2.2.2 The $m + \mathbf{b}$ Model

In addition to the overall average grade, some classes have higher averages than others. Here, we introduce an additional parameter per class, b_j, which produces the following criterion function:

$$J(m, \mathbf{b}) = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_i^j - m - b_j\right)^2 \left\{x_i^j \text{ is known}\right\} + \frac{\lambda}{2} \sum_{j=1}^{N} \left(b_j\right)^2, \tag{2.5}$$

where $\hat{x}_i^j = m + b_j$, λ is a tuning parameter to help control overfitting, and $\mathbf{b}$ is a row vector of class parameters:

$$\mathbf{b} = \begin{bmatrix} b_1, & b_2, & \ldots, & b_N \end{bmatrix}. \tag{2.6}$$

Once we compute m using Equation 2.4, b_j is the average grade for class j minus m:

$$b_j = \frac{1}{G^j + \lambda} \sum_{i=1}^{M} x_i^j \left\{x_i^j \text{ is known}\right\} - m, \tag{2.7}$$

where G^j is the number of known grades for class j. Figure 2.3 shows the model residuals. Here, model residuals decrease from left to right, reducing the errors due to class difficulty when compared to Figure 2.2.

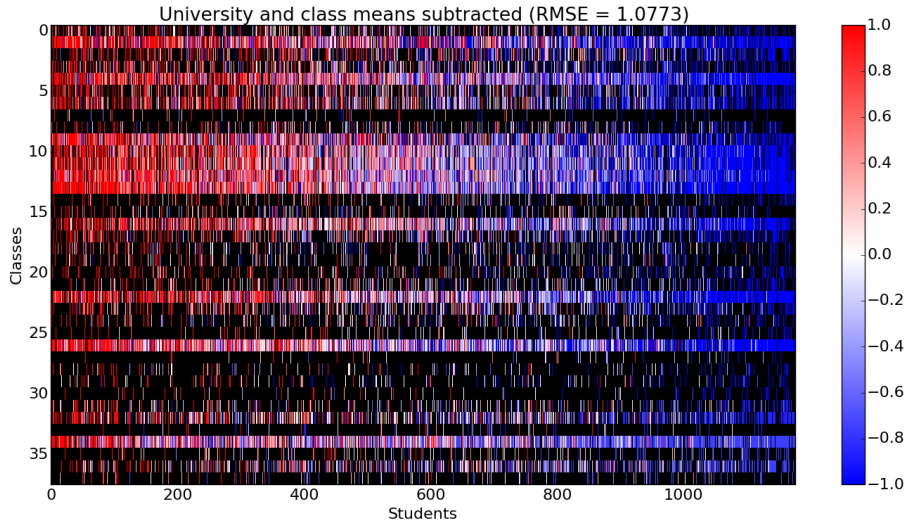

Figure 2.3: Heatmap of residuals for the m + **b** model

2.2.3 The $m + \mathbf{a}$ Model

Some students perform better than others and we introduce an additional parameter per student, a_i, which produces the following criterion function:

$$J(m, \mathbf{a}) = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_i^j - m - a_i\right)^2 \left\{x_i^j \text{ is known}\right\} + \frac{\lambda}{2} \sum_{i=1}^{M} \left(a_i\right)^2, \tag{2.8}$$

where $\hat{x}_i^j = m + a_i$, λ is still the tuning parameter, and $\mathbf{a}$ is a column vector of student parameters:

$$\mathbf{a} = \begin{bmatrix} a_1, & a_2, & \ldots, & a_M \end{bmatrix}^T. \tag{2.9}$$

Once we compute m using Equation 2.4, a_i is the average grade for student i minus m:

$$a_i = \frac{1}{G_i + \lambda} \sum_{j=1}^{N} x_i^j \left\{ x_i^j \text{ is known} \right\} - m, \tag{2.10}$$

where G_i is the number of known grades for student i. Figure 2.4 shows the model residuals. Here, model residuals decrease from top to bottom, reducing the errors due to differences in student performance when compared to Figure 2.2. Some of the lower residuals appear on the right side of the figure for students with higher averages.

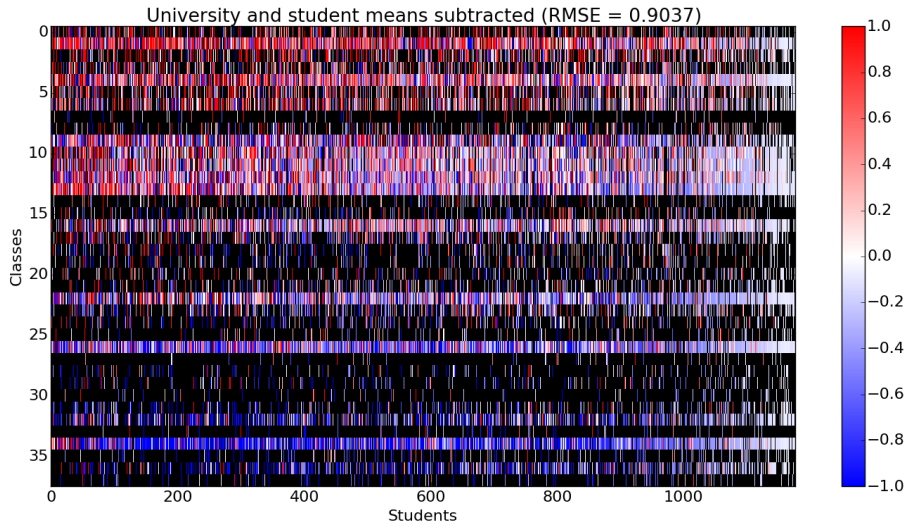

Figure 2.4: Heatmap of residuals for the m + **a** model

2.2.4 The $m + \mathbf{a} + \mathbf{b}$ Model

A better model might combine the class and student parameters to reduce the residuals. We combine m, $\mathbf{a}$, and $\mathbf{b}$, producing the following criterion function:

$$J(m, \mathbf{a}, \mathbf{b}) = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_i^j - m - a_i - b_j\right)^2 \left\{x_i^j \text{ is known}\right\} + \frac{\lambda}{2} \sum_{i=1}^{M} (a_i)^2 + \frac{\lambda}{2} \sum_{j=1}^{N} (b_j)^2, \tag{2.11}$$

where $\hat{x}_i^j = m + a_i + b_j$ and λ is the tuning parameter. We iterate between solving for m, $\mathbf{a}$, and $\mathbf{b}$ while holding the other parameters constant. This allows us to infer that a course with a high average might be the result of the high-average students who take it rather than its lack of difficulty. Specifically, we iterate between updating m, $\mathbf{a}$, and $\mathbf{b}$ in the following equations:

$$m \leftarrow \frac{1}{G} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_i^j - a_i - b_j\right) \left\{x_i^j \text{ is known}\right\} \tag{2.12}$$

$$a_i \leftarrow \frac{1}{G_i + \lambda} \sum_{j=1}^{N} \left(x_i^j - b_j - m\right) \left\{x_i^j \text{ is known}\right\} \tag{2.13}$$

$$b_j \leftarrow \frac{1}{G^j + \lambda} \sum_{i=1}^{M} \left(x_i^j - a_i - m\right) \left\{x_i^j \text{ is known}\right\}. \tag{2.14}$$

After incorporating both student and class parameters, Figure 2.5 shows the model residuals. The upper part of the figure (classes with lower averages) maintains a left-to-right

decreasing residual pattern. However, the lower part (classes with higher averages) does not. In addition, students with higher averages (on the left edge) tend to be underestimated, along with those in the lower-left. This suggests that there are different types of students and different types of classes that could be used to make better estimates. Nevertheless, this model fits better than any of the preceding models with a root mean squared error of 0.85.

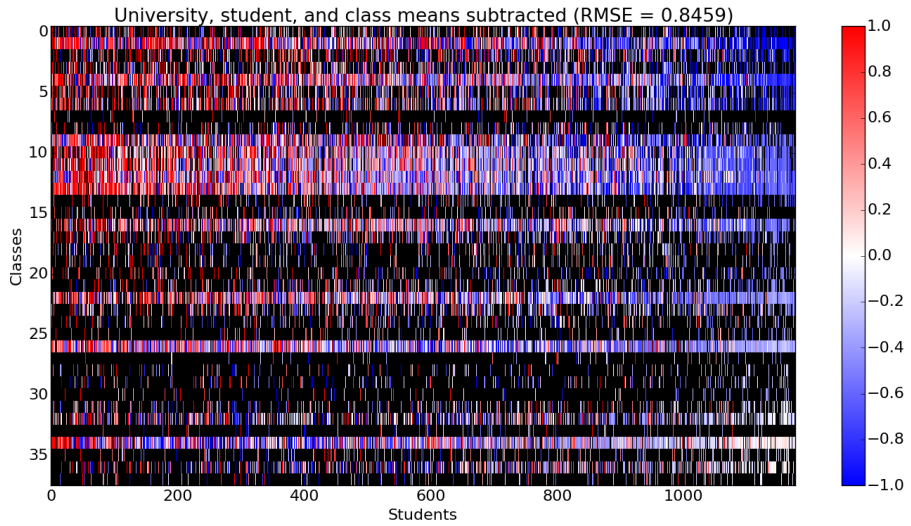

Figure 2.5: Heatmap of residuals for the m + **a** + **b** model

2.2.5 Alternating Least Squares

To address the remaining structure in the model residual, we attempt to factorize the residual matrix into a small number of components. For example, we perform a singular value decomposition (SVD) on the residual for the $m + \mathbf{a} + \mathbf{b}$ model. Although standard

software libraries provide the SVD for full matrices, they do not handle matrices like ours with mostly missing values. This complicates the estimation of singular vectors.

We employ an alternating least squares (ALS) approach to factorize the residual matrix, $\mathbf{Y} = \mathbf{X} - m - \mathbf{a} - \mathbf{b}$. Specifically, we factorize $\mathbf{Y}$ into the following [4]:

$$\mathbf{Y} \approx \mathbf{UV} \tag{2.15}$$

Specifically, The $M \times N$ matrix $\mathbf{Y}$ is approximated by the product of an $M \times P$ matrix $\mathbf{U}$ and a $P \times N$ matrix $\mathbf{V}$:

$$\mathbf{U} = \begin{bmatrix} u_1^1 & u_1^2 & \dots & u_1^P \\ u_2^1 & u_2^2 & \dots & u_2^P \\ \vdots & \vdots & \ddots & \vdots \\ u_M^1 & u_M^2 & \dots & u_M^P \end{bmatrix} \qquad \mathbf{V} = \begin{bmatrix} v_1^1 & v_1^2 & \dots & v_1^N \\ v_2^1 & v_2^2 & \dots & v_2^N \\ \vdots & \vdots & \ddots & \vdots \\ v_P^1 & v_P^2 & \dots & v_P^N \end{bmatrix}, \tag{2.16}$$

where P is the number of components chosen for the model. If we consider each student as a $1 \times N$ vector of grade residuals, each row of $\mathbf{U}$ represents the same information in a compressed P-dimensional vector. Similarly, if a class is represented as a $M \times 1$ residual vector, its corresponding column of $\mathbf{V}$ contains its compressed representation. In this P-dimensional space, the dot-product between students and classes provides a measure

of affinity. Positive dot-products add to the model's estimated grade for the student in that class.

U and **V** are updated concurrently with m, **a**, and **b** until convergence. For full matrices, the update for **U** and **V** would be the following:

$$\mathbf{U} \leftarrow \mathbf{YV}\left(\mathbf{V}^T\mathbf{V}\right)^{-1} \tag{2.17}$$

$$\mathbf{V} \leftarrow \left(\mathbf{U}^T\mathbf{U}\right)^{-1}\mathbf{U}^T\mathbf{Y}. \tag{2.18}$$

However, the majority of elements in **Y** are missing values, so another approach must be used.

2.2.6 Alternating Least Squares with Missing Values

ALS can also be used with data sets that have missing values. The items in **Y** are only used if they are known. Only the classes a student takes are used to update **U**, and only the data for students who have taken a specific class are used to update **V** [4]. The product of these two matrices gives an approximation of all values, given the known values in **Y**.

Instead of updating the entire $\mathbf{U}$ or $\mathbf{V}$ matrix at once, we can update each row or column at a time [4]. For example, the update for each row of $\mathbf{U}$ is the following:

$$\mathbf{u}_i \leftarrow \tilde{\mathbf{y}}_i \tilde{\mathbf{V}} \left(\tilde{\mathbf{V}}^T \tilde{\mathbf{V}} \right)^{-1}, \tag{2.19}$$

where $\mathbf{u}_i$ is the ith row of $\mathbf{U}$, $\tilde{\mathbf{y}}_i$ is the ith row of $\mathbf{Y}$ including only the known values, and $\tilde{\mathbf{V}}$ is a matrix containing the columns of $\mathbf{V}$ corresponding to the classes student i has taken. The update for each column of $\mathbf{V}$ is the following:

$$\mathbf{v}^j \leftarrow \left(\tilde{\mathbf{U}}^T \tilde{\mathbf{U}} \right)^{-1} \tilde{\mathbf{U}}^T \tilde{\mathbf{y}}^j, \tag{2.20}$$

where $\mathbf{v}^j$ is the jth column of $\mathbf{V}$, $\tilde{\mathbf{y}}^j$ is the jth column of $\mathbf{Y}$ containing only the students who have taken class j, and $\tilde{\mathbf{U}}$ is a matrix containing the rows of $\mathbf{U}$ corresponding to the students who have taken class j.

2.2.7 Principal Components Analysis (PCA)

Principal component analysis can be applied to data with missing values by alternately estimating the class means and the matrix factors such that $\mathbf{X} \approx \mathbf{UV} + \mathbf{b}$ [9]. For estimating student grades, we additionally remove the student means, producing the model by Sweeney et al. [5]: $\hat{\mathbf{X}} \approx m + \mathbf{a} + \mathbf{b} + \mathbf{UV}$. To infer the parameters of this model, we minimize the following criterion function with respect to each parameter in

turn:

$$J(m, \mathbf{a}, \mathbf{b}, \mathbf{U}, \mathbf{V}) = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}\left(x_i^j - m - a_i - b_j - \mathbf{u}_i\mathbf{v}^j\right)^2 \left\{x_i^j \text{ is known}\right\} + \frac{\lambda}{2}\sum_{i=1}^{M}(a_i)^2 + \frac{\lambda}{2}\sum_{j=1}^{N}(b_j)^2 + \frac{\lambda}{2}\sum_{i=1}^{M}\sum_{p=1}^{P}(u_i^p)^2 + \frac{\lambda}{2}\sum_{p=1}^{P}\sum_{j=1}^{N}\left(v_p^j\right)^2, \quad (2.21)$$

where $\hat{x}_i^j = m + a_i + b_j + \mathbf{u}_i\mathbf{v}^j$ and λ is the tuning parameter to help reduce the amount of overfitting. The heatmap of residuals for the PCA model with 1 component is shown in Figure 2.6. The colors in this heatmap are lighter than the previous figures, showing that many grades were better estimated.

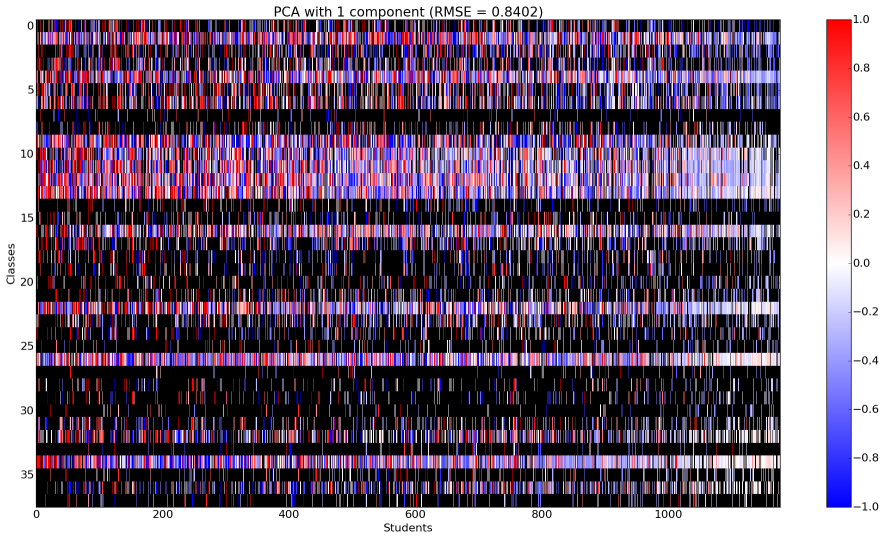

Figure 2.6: Heatmap of residuals for the PCA_1 model

2.3 Inferring the Parameters for Each Model

The criterion function can be minimized with respect to one parameter at a time by holding the others constant. By setting its partial derivative to zero, we can solve for one parameter's value to minimize the cost. We use the same algorithm for each model, initializing $\mathbf{V}$ to small random values and all other parameters to zero. Then, we update the parameters sequentially and iterate until convergence. If a particular parameter is not part of the model, we do not update it.

Below are the five update rules. Again, $\tilde{\mathbf{y}}^j$ contains the column vector of residuals for the jth class and $\tilde{\mathbf{U}}_j$ contains the P-dimensional row vectors for the students who have taken it. $\tilde{\mathbf{y}}_i$ contains the row vector of residuals for the ith student and $\tilde{\mathbf{V}}_i$ contains the column vectors for the classes they took. Also note that N is the number of classes, M is the number of students, and $\mathbf{I}$ is the $P \times P$ identity matrix.

$$m \leftarrow \frac{1}{G} \sum_{i=1}^{M} \sum_{j=1}^{N} \left(x_i^j - a_i - b_j - \mathbf{u}_i \mathbf{v}^j\right) \left\{x_i^j \text{ is known}\right\} \tag{2.22}$$

$$a_i \leftarrow \frac{1}{G_i + \lambda} \sum_{j=1}^{N} \left(x_i^j - m - b_j - \mathbf{u}_i \mathbf{v}^j\right) \left\{x_i^j \text{ is known}\right\} \tag{2.23}$$

$$b_j \leftarrow \frac{1}{G^j + \lambda} \sum_{i=1}^{M} \left(x_i^j - m - a_i - \mathbf{u}_i \mathbf{v}^j\right) \left\{x_i^j \text{ is known}\right\} \tag{2.24}$$

$$\mathbf{Y} \leftarrow \mathbf{X} - m - \mathbf{a} - \mathbf{b} \tag{2.25}$$

$$\mathbf{u}_i \leftarrow \tilde{\mathbf{y}}_i \tilde{\mathbf{V}}_i^T \left(\tilde{\mathbf{V}}_i \tilde{\mathbf{V}}_i^T + \lambda \mathbf{I}\right)^{-1} \tag{2.26}$$

$$\mathbf{v}^j \leftarrow \left(\tilde{\mathbf{U}}_j^T \tilde{\mathbf{U}}_j + \lambda \mathbf{I}\right)^{-1} \tilde{\mathbf{U}}_j^T \tilde{\mathbf{y}}^j \tag{2.27}$$

These updates are performed 1,000 times or until convergence as shown in Algorithm 1.

Algorithm 1: Pseudocode for the PCA model

Result: m, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{U}$, and $\mathbf{V}$
Initialize m to be zero;
Initialize $\mathbf{a}$ to be a $M \times 1$ vector of zeroes;
Initialize $\mathbf{b}$ to be a $1 \times N$ vector of zeroes;
Initialize $\mathbf{U}$ to be a $M \times P$ matrix of zeroes;
Initialize $\mathbf{V}$ to be a $P \times N$ matrix of small random values;
error $= \infty$;
while *True* **do**
    Update m according to Equation 2.22;
    for $i \leftarrow 1$ **to** M **do**
        Update a_i according to Equation 2.23;
    end
    for $j \leftarrow 1$ **to** N **do**
        Update b_j according to Equation 2.24;
    end
    $\mathbf{Y} = \mathbf{X} - m - \mathbf{a} - \mathbf{b}$;
    for $i \leftarrow 1$ **to** M **do**
        Update $\mathbf{u_i}$ according to Equation 2.26;
    end
    for $j \leftarrow 1$ **to** N **do**
        Update $\mathbf{v_j}$ according to Equation 2.27;
    end
    Normalize each row of V so it has unit length, and scale the corresponding column of U accordingly;
    Calculate the root mean squared error (RMSE) of the known values;
    Break if RMSE $< 1e-05$ or the percent difference in the previous and current error $\leq 1e-05$;
end
Orthogonalize the columns of $\mathbf{U}$ and rows of $\mathbf{V}$ using SVD;

After fitting, the SVD is used to produce orthogonal axes in $\mathbf{U}$ and $\mathbf{V}$.

2.4 Model Evaluation

Each model has multiple parameters that it must learn. In addition, all but one of the described models has a tuning parameter, λ that controls against overfitting. Specifically, we consider powers of ten between 0.0001 and 1,000 [10]. We vary λ and select the one that produces the lowest test error in a grid search [8].

Nested cross-validation is used to find the best λ and compare the models. In this approach, ten stratified folds are used, similar to cross-validation. However, the training set is then split into nine folds: eight are used to train the model with an experimental λ, while the ninth fold gauges how well the model estimates unseen data. If we come across students that have not been seen, then we only use m and $\mathbf{b}$ to estimate their grades. Using the best λ, we train on all nine folds and test on the tenth fold. Once again, the RMSE provides a number detailing how well each model performed.

Chapter 3 - Results

We compare 14 total models: m, $m + \mathbf{a}$, $m + \mathbf{b}$, $m + \mathbf{a} + \mathbf{b}$, and PCA with between 1 and 10 components. In addition, all but one model have a λ parameter that must be tuned. These models are compared using two data sets: the computer science data set and the more general data set.

3.1 Testing the Approach

To check whether or not the model behaves as expected, we generate synthetic data for each of the models where m, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{U}$, and $\mathbf{V}$ are known. The variables are drawn from a uniform normal distribution whose range of values is [-0.5, 0.5), except m which is between 2 and 3. Each data set contains 1,000 rows and 30 columns. Each model is fit with its specific data set and scored with the same data. Table 3.1 shows the RMSE values of the different models.

Table 3.1: RMSE Values for the Random Models

Model	**RMSE Mean**
m	0
$m + \mathbf{b}$	3.7551×10^{-15}
$m + \mathbf{a}$	0
$m + \mathbf{a} + \mathbf{b}$	3.7370×10^{-15}
PCA_1	3.1250×10^{-15}
PCA_2	1.6676×10^{-15}
PCA_3	2.1116×10^{-15}
PCA_4	1.3782×10^{-14}
PCA_5	1.6045×10^{-13}
PCA_6	1.3964×10^{-11}
PCA_7	5.0431×10^{-12}
PCA_8	2.6321×10^{-12}
PCA_9	4.4230×10^{-11}
PCA_{10}	2.5892×10^{-10}

Each model learns the variables such that the RMSE is near zero. Since the RMSE is near zero, each variable is learned with little error. In the case of the m and $m + \mathbf{a}$ models, the RMSE is zero. This shows the models is capable of inferring users and classes, given complete control of the data.

3.2 Results for the Computer Science Data Set

First, we consider every model and choice of λ, and compare them based on cross-validation performance. We use ten fold stratified cross-validation, which preserves the number of grades per student across folds. Specifically, we use sklearn's GridSearchCV function [11]. Below is a heatmap detailing the results:

Figure 3.1: Heatmap of RMSEs for the PCA model

In this figure, the simplest models are in the bottom-left portion of the figure, while the most complex models are in the top-right. Simpler models have larger λ values. The simplest models are the m models (the first column of values). The mb column is for the $m + \mathbf{b}$ models, while the ma column is for the $m + \mathbf{a}$ models. The 0 column is for the the $m + \mathbf{a} + \mathbf{b}$ models, while numbers 1-10 indicate the number of components for the PCA models. The darker region on the left suggests that these models are too simple, while the darker region in the top-right indicates models that are too complex. There is a band of lighter colors that have the lowest RMSEs.

The PCA_8 model where $\lambda = 10$ is the one that is used for the modeling of computer science students. Since this model has the lowest RMSE of any model presented,

it seems that having components helps estimate a student's grade. This suggests that students may have an affinity for certain classes, which are shown by these components.

3.3 Interpreting Components of Computer Science Grades

After training the models, they have now learned students and classes in a P-dimensional subspace. The best model was the PCA_8 model, so each student and each class is now represented by eight distinct values.

The **U** matrix shows how a student is projected onto a particular axis, depending on the column of **U**. The value a student gets in this column relates to the same component of **V**, which is a row. Positive students and positive classes, as well as negative students and negative classes, give a grade boost for that component. If the student's value and class's value are not the same sign, then it is a detriment to the estimated grade. Figure 3.2 shows the distribution of the students in **U**'s all eight components as a histogram.

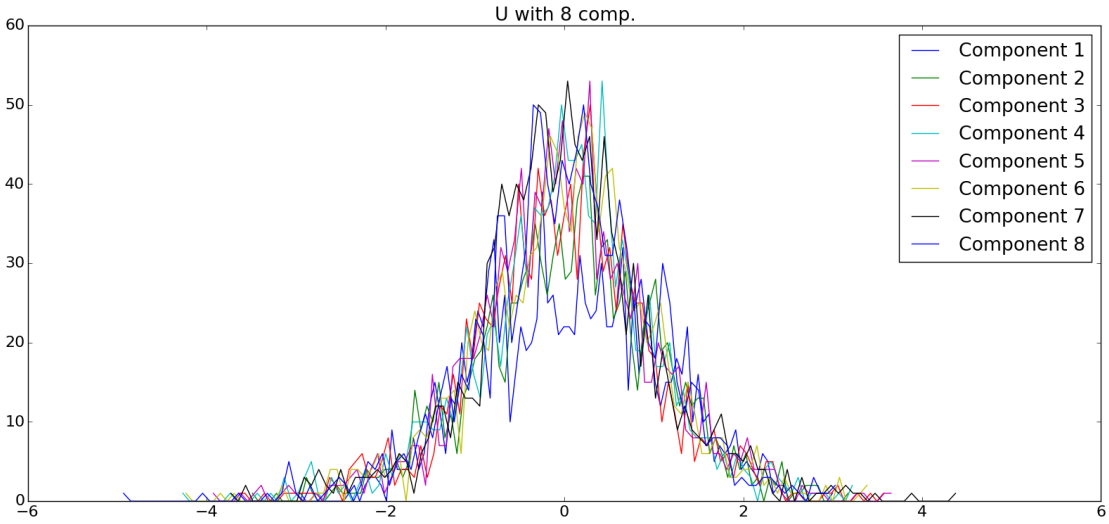

Figure 3.2: The eight **U** components in the PCA_8 model

The mean for each of the curves is near zero, while the standard deviation is one. Components 3, 5, and 7 slightly favor positive values, while the sixth component favors negative values. Many students have values near zero, meaning that these components have little effect on the estimated grade for that student. However, a student that is one standard deviation above the mean would get their estimate increased by the height of the class's bar in the component's corresponding V figure. These figures show the representation of each class in the corresponding component. The first component of **V** is shown in Figure 3.3.

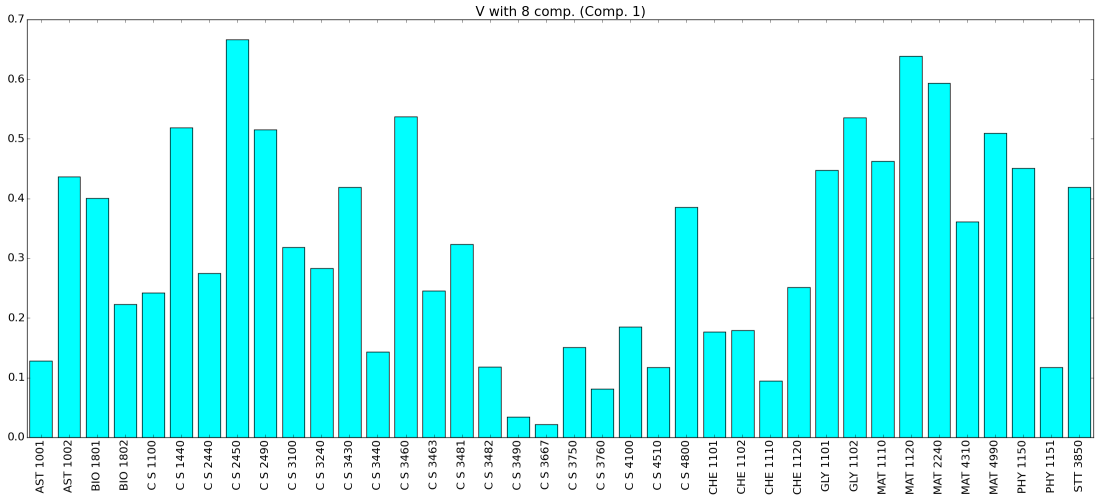

Figure 3.3: First **V** component in the PCA model

An interesting feature in Figure 3.3 is that Programming Languages (CS 3490) and Software Engineering (CS 3667) are near zero. This indicates that this component has little to no effect on the grades earned in these courses by the students who have taken them. The largest values are for Intro to Computer Systems (CS 2450), Calculus 2 (MAT 1120), and Linear Algebra (MAT 2240). Each of these classes are near or above 0.6. All classes in this component are positive. However, the second component, shown in Figure 3.4, shows some classes with negative values.

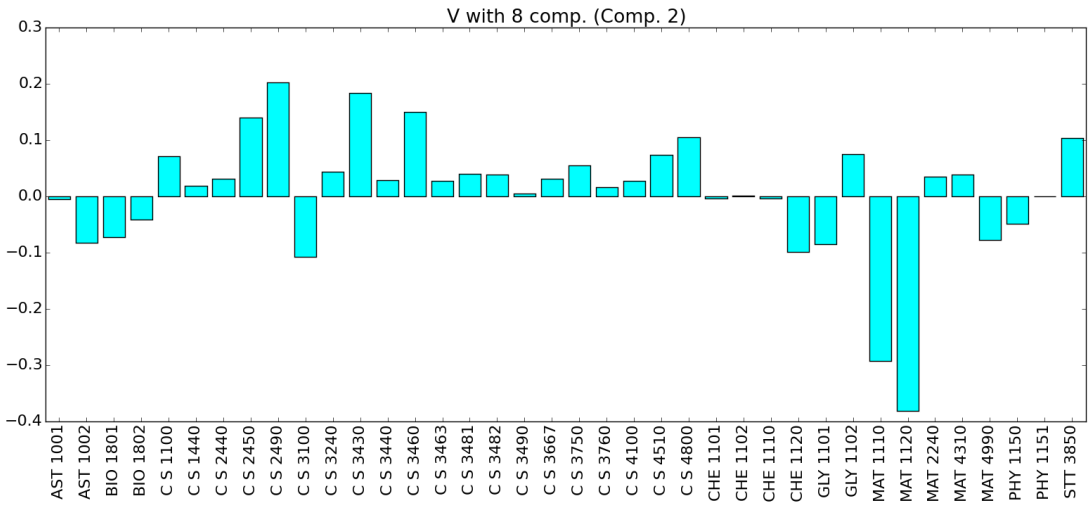

Figure 3.4: Second **V** component in the PCA model

In this figure, Calculus 1 (MAT 1110) and Calculus 2 (MAT 1120) are both below -0.25, while the next lowest class is Junior Seminar (CS 3100) close to -0.1. The largest two values shown in this graph are for Theoretical Computer Science (CS 2490) and Database (CS 3430).

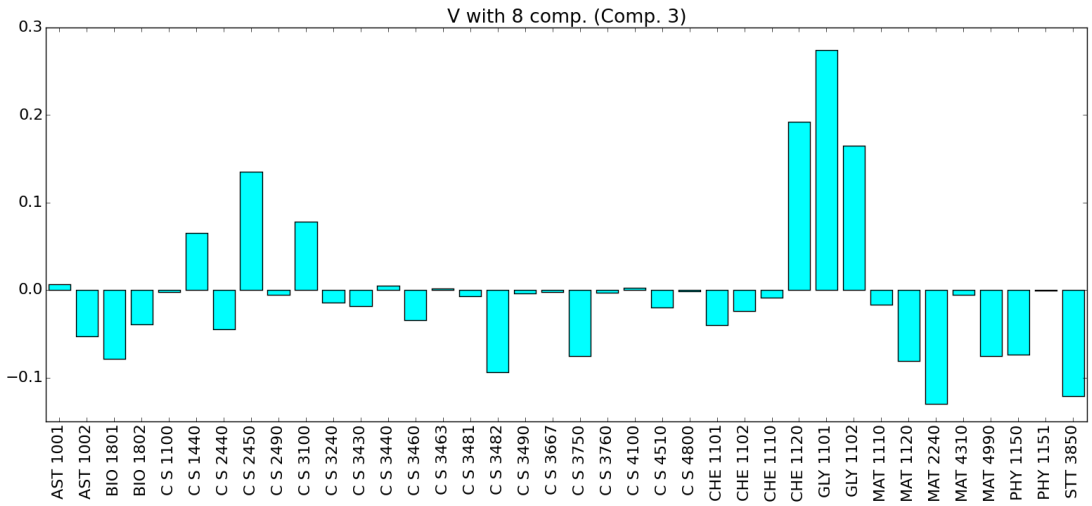

Figure 3.5: Third **V** component in the PCA model

Intro to Physical Geology (GLY 1101) and CHE 1120 are both positive in this figure, while MAT 2240 and Statistical Data Analysis 1 (STT 3850) are near -0.1. STT 3850 becomes positive in Figure 3.6.

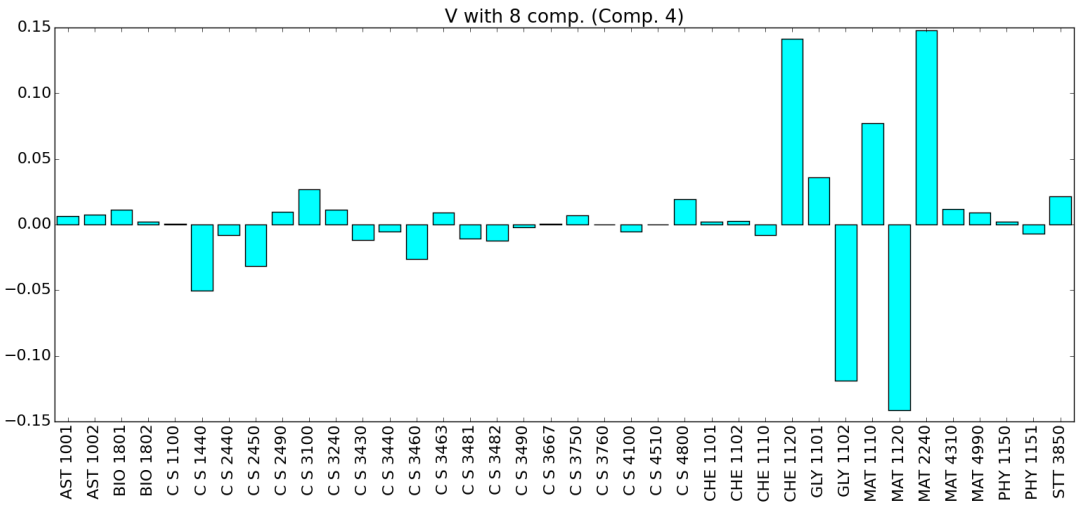

Figure 3.6: Fourth **V** component in the PCA model

CHE 1120 and MAT 2240 are positive in this figure. Calculus 2 and Intro to Historical Geology (GLY 1102) are the only two classes below -0.1. This trend continues in Figure 3.7.

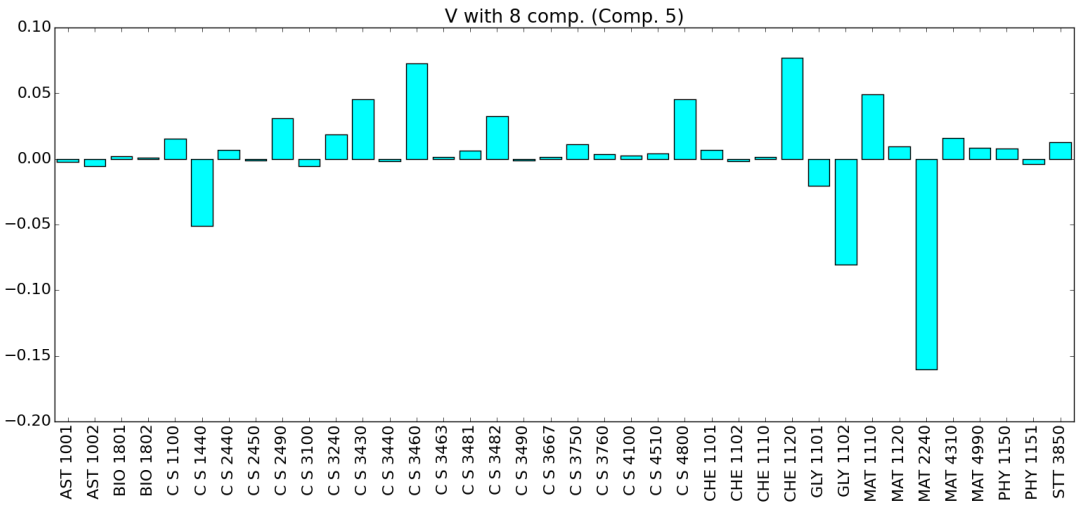

Figure 3.7: Fifth **V** component in the PCA model

The largest positive values in component five are CS 3490 and CHE 1120. The only three classes below -0.05 are Computer Science 1 (CS 1440), GLY 1102, and MAT 2240. CS 1440 also has one of the most negative values in Figure 3.8.

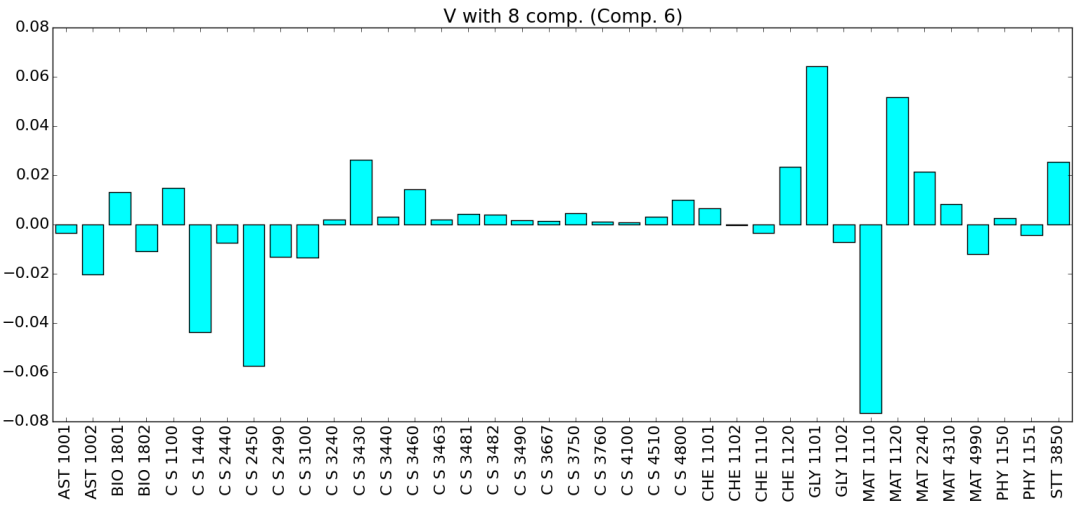

Figure 3.8: Sixth **V** component in the PCA model

Many of the beginning computer science courses, barring CS 1100, are negative in this dimension. The Astronomy courses, as well as MAT 1110 and Numerical Linear Algebra (MAT 4990), are also negative. However, many classes are either negative or near zero. This trend continues in Figure 3.9.

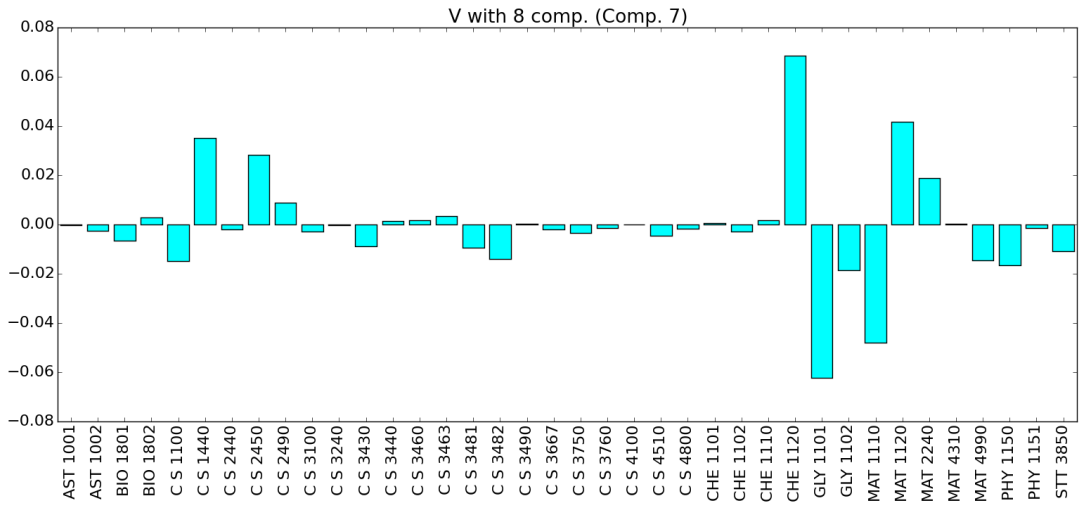

Figure 3.9: Seventh **V** component in the PCA model

Most classes are either negative or close to zero in this component. The GLY courses and MAT 1110 are among the most negative. CS 1440, CHE 1120, and MAT 1120 are among the most positive. Many classes are far from zero in Figure 3.10.

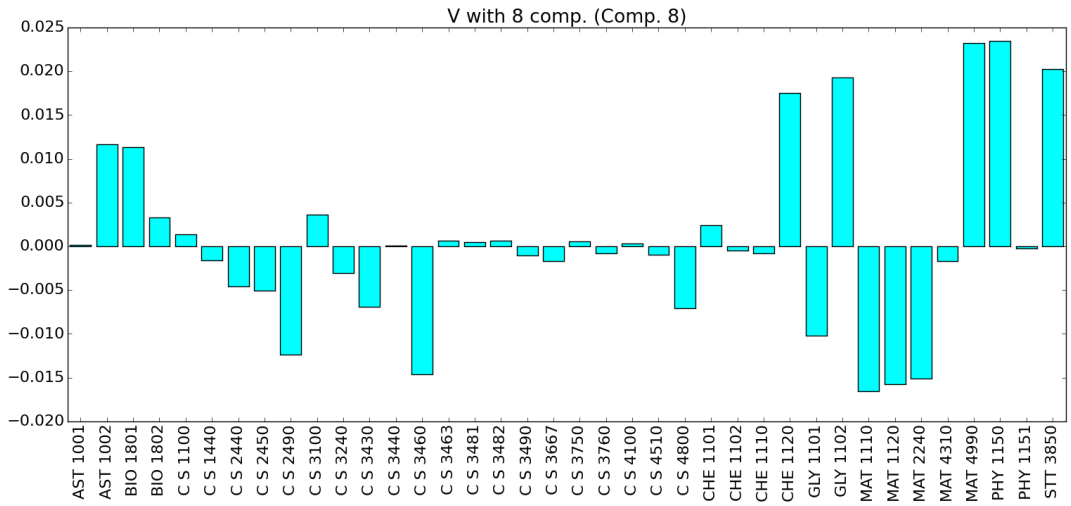

Figure 3.10: Eighth **V** component in the PCA model

There are many classes that have positive values, including Physics 1 (PHY 1150) and STT 3850. All of the math classes, except MAT 4990, are negative in this dimension.

After observing the components and using a grid search to find the optimal parameters for this data set, a nested cross-validation is done to see which model was the best at estimating future performance. Below is a table comparing the models.

Table 3.2: RMSE Values for the Optimal CS Models

Model	**RMSE Mean**	**RMSE St. Dev.**
m	1.1191	0.0194
$m + \mathbf{b}$	1.0773	0.0193
$m + \mathbf{a}$	0.9055	0.0172
$m + \mathbf{a} + \mathbf{b}$	0.8770	0.0175
PCA_1	0.8468	0.0148
PCA_2	0.8398	0.0152
PCA_3	0.8376	0.0154
PCA_4	0.8381	0.0143
PCA_5	0.8369	0.0149
PCA_6*	0.8348	0.0153
PCA_7*	0.8334	0.0149
PCA_8*	0.8335	0.0150
PCA_9*	0.8338	0.0154
PCA_{10}*	0.8334	0.0151

* Models that are not significantly different from the top performing model, as determined by a paired t-test resulting in a p-value > 0.05

The PCA_7 and PCA_{10} models have the lowest RMSE by 0.0001 over the PCA_8 model. Every PCA model performs better than the $m + \mathbf{a} + \mathbf{b}$ model, whose RMSE is 0.0302 higher than any of the PCA models.

3.4 Results for the More General Data Set

Similarly to Section 3.2, ten folds are used to evaluate the models for the data set that involves the CS classes, as well as classes in the most common courses that are not already in the list. This changes the data set from 1,177 students by 38 classes to 9,554 students by 70 classes. Below is a heatmap of the RMSE values for all the models.

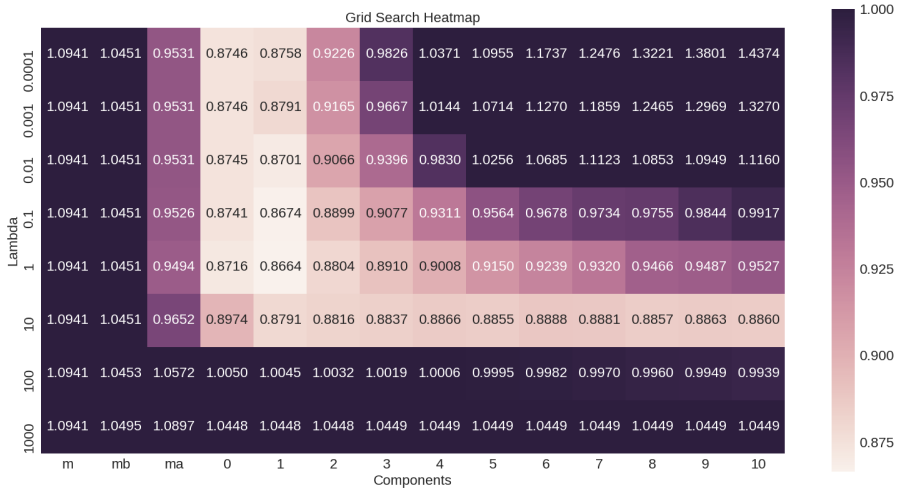

Figure 3.11: Heatmap of RMSEs for the PCA model

The models for this data set act similarly to those in the CS data set. However, there are several λ values that are close with a lower number of components, specifically 0.1 and 1. The other models optimized at different positions. The lowest RMSE was observed to be 0.8641 using the PCA_1 model, which is an improvement of 0.0055 over not using PCA.

3.5 Interpreting Components of More General Grades

The PCA_1 model has now learned students and classes in a one-dimensional subspace. **U** and **V** are used to find potential student-class interactions in this one-dimensional

subspace. The distribution of the **U** matrix for the single component is shown in Figure 3.12.

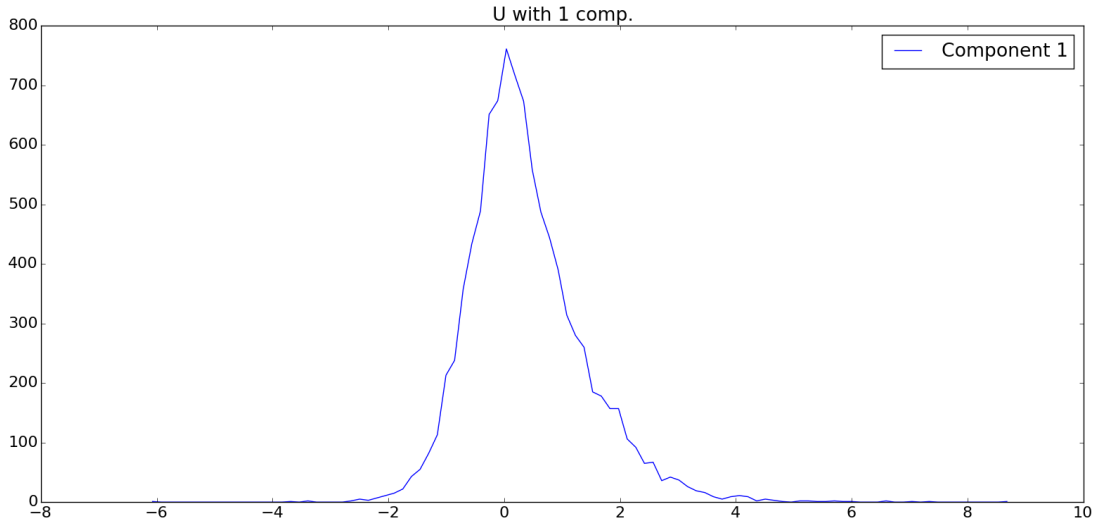

Figure 3.12: First **U** component in the PCA model

This distribution has a mean of 0.3790, which would mean that many students are slightly positive when it comes to favoring certain classes in the only component of this model. The standard deviation is once again 1, while the positive tail is longer than the negative tail. There are a few outliers, which are shown in the tails reaching -6 and 8.

The **V** matrix contains all 70 classes and how they are projected in the single-dimension subspace the model found. Graphing these values visually describes how the classes relate to each other. For example, if one class is positive in **V** and another is negative, then doing well in one would lower the grade received in the other. Figure 3.13 shows these classes and their values in **V**.

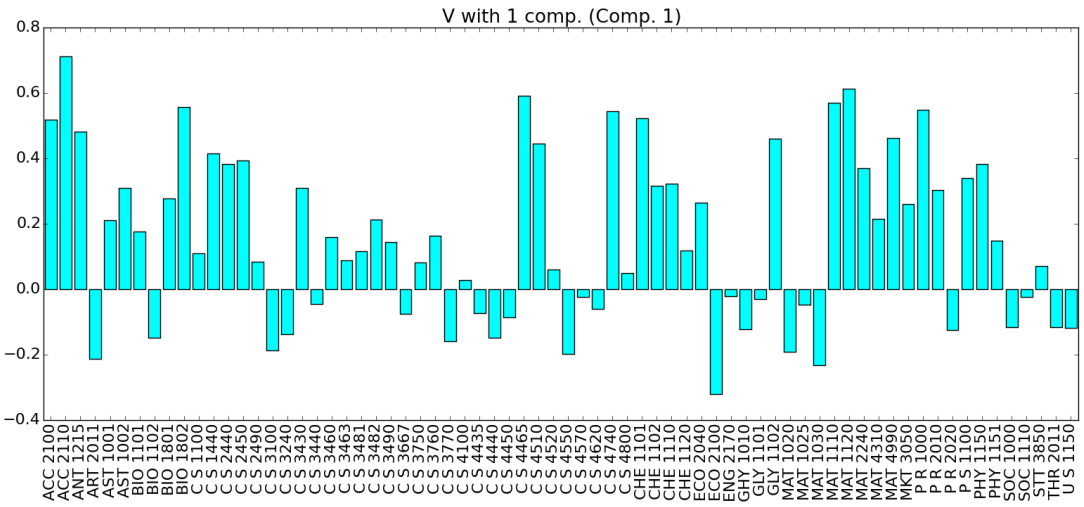

Figure 3.13: First **V** component in the PCA model

Most courses are positive in this component, including most computer science and math classes. However, there are several classes that are negative, which include ART 2011, CS 3100, ECO 2100, and MAT 1030.

After observing the components and using a grid search to find the optimal parameters for this data set, a nested cross-validation is done to see which model was best at estimating future grades. Below is a table comparing the models.

Table 3.3: RMSE Values for the Optimal General Models

Model	**RMSE Mean**	**RMSE St. Dev.**
m	1.0944	0.0044
$m + \mathbf{b}$	1.0455	0.0055
$m + \mathbf{a}$	0.9532	0.0081
$m + \mathbf{a} + \mathbf{b}$	1.0054	0.0052
PCA_1	1.0049	0.0052
PCA_2	1.0035	0.0052
PCA_3	1.0022	0.0051
PCA_4	1.0009	0.0051
PCA_5	0.9997	0.0051
PCA_6	0.9985	0.0051
PCA_7	0.9973	0.0051
PCA_8	0.9962	0.0051
PCA_9	0.9951	0.0051
PCA_{10}	0.9941	0.0051

With this data set, the best model was $m + \mathbf{a}$. This suggests that there is not enough information to properly learn characteristics about the classes. Nearly half the classes in this data set are considered general education, which may also add additional levels of complexity for the models to interpret.

Chapter 4 - Conclusion and Future Work

The models used in this thesis were able to provide insight about student-class interactions. The models using Principal Components Analysis achieved a lower RMSE than those without PCA. Knowing this can help the understanding of the types of students and classes that are in the computer science undergraduate curriculum at Appalachian State University. The model with the best performance was the PCA_8 model for the computer science data set. This model's first component shows that the most positive class is CS 2450, while the smallest value is for CS 3667. This component could show the variance of the classes after the university mean, user means, and class means have been removed. Students who are a standard deviation above the average would get their estimate significantly boosted for CS 2450, but not for CS 3667. The second dimension could represent programming classes in contrast to classes that do not involve programming. Most computer science classes are positive, as is STT 3850, which uses the R statistical programming language. Classes that do not rely as much on programming skills, like MAT 1110 and GLY 1101, are negative in this component. The third axis supplied by this model may relate to analytical classes versus hands-on laboratory classes. CHE 1120 and GLY 1101 are both classes that require interaction with the material in labs. CS 2450 is similar in this manner, as this class introduces assembly language. Classes, such

as MAT 1120 and MAT 2240, are more analytical in the fact that students have to apply the concepts learned through assignments, rather than labs.

Adding general education classes into the data set introduced over 8,000 new students and 32 classes. The overall error for this data set was higher than using only computer science data. However, the models using PCA were unable to estimate grades as accurately as the $m + \mathbf{a}$ model, possibly because many of the general education classes are from different disciplines. Many students only took one or two computer science courses, but took at least ten general education classes. PCA has the potential to point out some features of students and classes, given the right data and circumstances. The results detailed in the previous chapter show that more research is needed to better understand the full potential of this approach for estimating grades.

4.1 Future Work

In the future, the models shown in this thesis should be attempted with a larger data set, in order to understand how well this approach works. These models could also be applied to other majors, such as Biology or Art. This would allow for the interpretation of other majors in order to learn the relationships between their classes and the students who take them.

The λ parameter could possibly be better tuned. Only models that used either 1 or 10 were able to estimate student grades effectively. If the values between 1 and 10 were used, there is a possibility that the grades would be better estimated.

Other machine learning algorithms may also be applicable to this problem. An exploration of these algorithms, and whether or not they could be applied to a problem similar to this, could aid in learning the different types of classes and students. One potential for improvement would be to change the focus of the components from being an axis to a vector. This would show how much of a particular component is present in the data, instead of having the possibility of containing the 'opposite' of that component.

Bibliography

[1] S. Aud, S. Wilkinson-Flicker, P. Kristapovich, A. Rathbun, X. Wang, and J. Zhang, "The Condition of Education 2013. NCES 2013-037.," *National Center for Education Statistics*, 2013.

[2] A. T. Chamillard, "Using Student Performance Predictions in a Computer Science Curriculum," in *ACM SIGCSE Bulletin*, vol. 38, pp. 260–264, ACM, 2006.

[3] J. Bennett and S. Lanning, "The Netflix Prize," in *Proceedings of KDD Cup and Workshop*, vol. 2007, p. 35, 2007.

[4] C. O'Neil and R. Schutt, *Doing Data Science*. O'Reilly Media, Inc., Oct. 2013.

[5] M. Sweeney, J. Lester, and H. Rangwala, "Next-Term Student Grade Prediction," in *2015 IEEE International Conference on Big Data*, pp. 970–975, Oct. 2015.

[6] A. Elbadrawy, A. Polyzou, Z. Ren, M. Sweeney, G. Karypis, and H. Rangwala, "Predicting Student Performance Using Personalized Analytics," *IEEE Computer Society*, vol. 49, no. 4, pp. 61–69, 2016.

[7] Z. Ren, X. Ning, and H. Rangwala, "Grade Prediction with Temporal Course-Wise Influence," in *The 10th International Conference on Educational Data Mining*, (Wuhan, China), 2017.

[8] S. Raschka, *Python Machine Learning*. Packt Publishing, 1st ed., Sept. 2015.

[9] A. Ilin and T. Raiko, "Practical Approaches to Principal Component Analysis in the Presence of Missing Values," *Journal of Machine Learning Research*, vol. 11, no. Jul, pp. 1957–2000, 2010.

[10] A. Holehouse, "Stanford Machine Learning," Mar. 2012. [Available at http://www.holehouse.org/mlclass/; accessed 2017-10-17].

[11] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[12] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating Collaborative Filtering Recommender Systems," *ACM Transactions on Information Systems*, vol. 22, pp. 5–53, Jan. 2004.

[13] J. Davidson, B. Liebald, J. Liu, *et al.*, "The YouTube Video Recommendation System," in *Proceedings of the Fourth ACM Conference on Recommender Systems*, pp. 293–296, ACM, 2010.

[14] O. Hinz and J. Eckert, "The Impact of Search and Recommendation Systems on Sales in Electronic Commerce," *Business & Information Systems Engineering*, vol. 2, pp. 67–77, Apr. 2010.

[15] G. Linden, B. Smith, and J. York, "Amazon.com Recommendations: Item-to-Item Collaborative Filtering," *IEEE Internet Computing*, vol. 7, pp. 76–80, Jan. 2003.

[16] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Matrix Factorization and Neighbor Based Algorithms for the Netflix Prize Problem," in *Proceedings of the 2008 ACM Conference on Recommender Systems*, pp. 267–274, ACM, 2008.

[17] R. Zhou, S. Khemmarat, and L. Gao, "The Impact of YouTube Recommendation System on Video Views," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, pp. 404–410, ACM, 2010.

[18] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, pp. 51 – 56, 2010.

[19] G. Rossum, "Python Reference Manual," tech. rep., Amsterdam, The Netherlands, The Netherlands, 1995.

[20] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open Source Scientific Tools for Python," 2001. [Available at http://www.scipy.org; accessed 2017-06-10].

[21] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy Array: A Structure for Efficient Numerical Computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[22] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[23] M. Waskom, O. Botvinnik, D. O'Kane, *et al.*, "Mwaskom/Seaborn: V0.8.1 (September 2017)," Sept. 2017. DOI: 10.5281/zenodo.883859.

[24] S. Lohr, "A $1 Million Research Bargain for Netflix, and Maybe a Model for Others," Sept. 2009. [Available at http://www.nytimes.com; accessed 2017-09-16].

Vita

Christopher Smith was born to Timmy and Kathy Smith in September 1993 in the state of North Carolina, United States of America. After attending Caldwell Community College and Technical Institute and graduating with an Associates in Science in May 2013, he entered Appalachian State University in August 2013 where he earned his Bachelor of Science in Computer Science in December 2015. Afterwards, he was accepted into the Master of Science in Computer Science program at Appalachian State University, where he graduated in December 2017.