CS 365 Project Final

Jae Hong Lee and Wyatt Napier
Boston University
jhonglee@bu.edu wnapier@bu.edu

Abstract

For our project milestone we're using Principal-Component Analysis (PCA) and Support Vector Machines (SVM) to predict housing prices based off of this dataset provided by Kaggle. This is our Google Colab notebook.

1 Introduction

1.1 Problem

As college students in Boston, housing prices are often on our minds, especially as we look for off-campus housing. Housing price is based on a variety of factors, but we want to ensure that we are getting the most value for our money. By creating a model to estimate housing prices based off of the attributes of a specific house, we will more objectively be able to determine if the housing we're looking at is truly worth its price. We will use PCA to do some preprocessing on our data and then use support vector machines and radial basis functions for our regression model.

1.2 Dataset

The dataset that we're going to use to train our model comes from Kaggle. There are 1460 rows of data which we will concatenate and do our own train/test split in order to optimize the ratio so as to not over or undertrain. This dataset is incredibly verbose as it has roughly 80 different columns. We can analyze the differences in results based on the selected attributes and diversify the combinations of attributes chosen to increase accuracy.

1.3 Resources

In our project we used Google Colab to write and run our code. On Google Colab, we used roughly 28.2 GB of disk space as well as 1.4 GB of RAM. We were most frequently connected to the CPU runtime since we could only use the T4 GPU for a limited amount of time. This was sufficient for running all of the code besides a small portion at the end where we attempt to compare the results of the model on the original pre-processed data with the results of the model on the PCA representation of the original pre-processed data. In this case, the runtime disconnnects before it completes. This typically occurs after at least 1.5 hours of running. We will elaborate on why that occurred in the results section of this paper. Some other tools that we relied on were Sci-Kit Learn documentation, ChatGPT to help debug our code, and the various textbooks we cite in the references section.

2 Principal Component Analysis

Principal Component Analysis (PCA) is a method of compressing the dimension of data to make learning easier and prediction more efficient. We want to go from representing the data in R^d to a significantly smaller space R^p by using p principal components from the eigendecomposition.

We want to approximate an input vector $\mathbf{x^{(n)}} \in R^d$ with basis vectors $\mathbf{v_1}, ..., \mathbf{v_d} \in R^d$ by fulfilling the following equation where each α is a representation coefficient: $\mathbf{x^{(n)}} \approx \sum_{i=1}^{p} \alpha_i \mathbf{v_i}$. To do so we need to find both the basis vectors $\{\mathbf{v_i}\}_{\mathbf{i=1}}^{\mathbf{P}}$ and coefficients $\{\alpha_i\}_{i=1}^{p}$. We do this using an eigendecomposition which is often derived from SVD or the definition of eignevalues and eigenvectors: $\mathbf{Au} = \lambda\mathbf{u} \rightarrow \mathbf{A} = \mathbf{U\Lambda U^{-1}}$.

For just one row of data we would need to solve this optimization problem: $(\hat{\alpha}, \hat{\mathbf{v}}) = argmin_{||\mathbf{v}||_2=1,\alpha}||\mathbf{x} - \alpha\mathbf{v}||^2$. Basically what this means is we want to find the coefficient and the basis vector such that we can minimize the distance of $\mathbf{x}$ from the scaled vector $\alpha\mathbf{v}$. To generalize this to the entire matrix we have the following equations:

$$\hat{\mathbf{v}} = argmin_{||\mathbf{v}||_2=1}E||\mathbf{X} - \alpha\mathbf{v}||^2 = argmax_{||\mathbf{v}||_2=1}\mathbf{v^T\Sigma v}$$

We want to maximize the variance of the data, while also minimizing the distance of the data from the basis vectors. These two problems happen to actually be the same problem. Since in this case, $\mathbf{\Sigma}$ is just the covariance matrix from $\mathbf{\Sigma} = E[\mathbf{X^T X}]$, we know that the $\hat{\mathbf{v}}$ has a solution $\mathbf{u_1}$ which is the first column of the eignevector matrix $\mathbf{U}$ assuming it is ordered by magnitude of corresponding eigenvalue.

Stepping back for a moment, we want to maximize the variance along the basis vectors which will also minimize the distance of each point from that line. Since we're using the covariance matrix the diagonal will just be the variance. When we do the eigendecomposition, the eigenvalues are then a measure of the variance so the magnitude of the eigenvalues corresponds to the variance of each.

To clarify, the eigendecomposition of the dxd matrix $\mathbf{\Sigma}$ will give us the eigenvector matrix $\mathbf{U}$ and the eigenvalue matrix $\mathbf{S}$. As part of PCA we want to use just p principle components (started with d components) so we just truncate $\mathbf{U}$ and $\mathbf{S}$ to use only the first p eigenvectors and eigenvalues. In doing so, we project the points $\{\mathbf{x^{(1)}}, ..., \mathbf{x^{(n)}}\}$ onto the space spanned by the first p eigenvectors of $\mathbf{U}$. We use the following equation for a single point and repeat for the entire dataset to compress it:

$$\mathbf{U_p^T x^{(n)}} = \alpha^{(\mathbf{n})}$$

Overall, the structure of PCA is to center the mean of the data on the origin and then find the eigendecomposition of the covariance matrix so that we can then do the reduction. In some applications the data is first scaled for each feature before applying SVD. PCA'a foundations lay in simple linear algebraic techniques such as SVD and eigendecomposition and as such it is a true testament to the power of linear algebra.

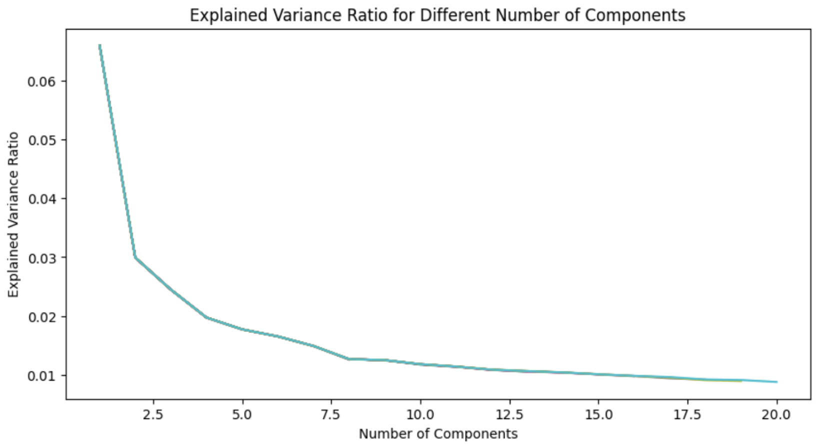


Figure 1: Explained Variance Ratio for Different Number of Components

When applying this to our dataset, we want to find the optimal number of principal components to maximize the explained variance ratio. The explained variance ratio equation is the percentage of the variance explained or captured by the selected component. In Figure 1 above, since the explained variance ratio has such a steep decay, only a small number of principal components are really needed. It is evident that the magnitude of the explained variance ratio after the 8th principal component seems to flatten out, so we chose to limit our PCA to 8 principal components to represent our preprocessed 271 columns.

3 Support Vector Machine

3.1 Decision Rule

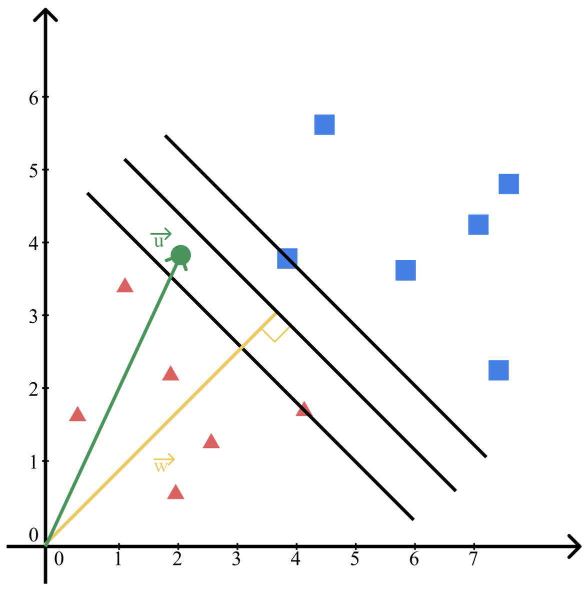


Figure 2: Decision Boundary

The space in the middle is drawn based on the widest path between the closest vectors (commonly called support vectors) of two different categories. Therefore, it's referred to as the widest street approach. In the method of dividing examples of two kinds: triangles and rectangles, this path is made as wide as possible. There is a vector $\vec{w}$ of some length. This $\vec{w}$ is perpendicular to the dividing center-line of the plane, and therefore the distance from the origin to the center-line is minimal. The distance of this vector is unknown.

And there is another unknown point. It is represented by $\vec{u}$ in the figure. It is important to determine whether this point is closer to a triangle or a rectangle. Now, by projecting this point onto the $\vec{w}$ vector, we can determine how many times it is proportional to the $\vec{w}$ vector. If the length of the projection of $\vec{u}$ onto $\vec{w}$ is closer to the center-line, it is classified as a triangle; otherwise, if it is further than the center-line, it is classified as a rectangle. The figure above $\vec{u}$ would be classified as a triangle.

The projection of $\vec{u}$ to $\vec{w}$ is the dot product of two vectors.

This is SVM's Decision Rule

$$\vec{w} \cdot \vec{u} \geq some\ constant\ c$$
$$if\ c = -b,\ \vec{w} \cdot \vec{u} + b \geq 0\ then\ Rectangle$$

3.2 Margin Constraints

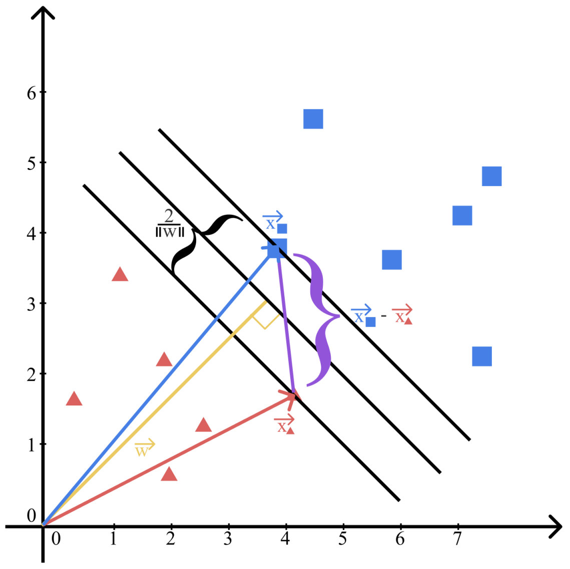
What conditions are needed for the constants b and vector $\vec{w}$? We only know that vector $\vec{w}$ is perpendicular to the central line. However, since $\vec{w}$ can be perpendicular to the central line regardless of its length, there are no constraints on $\vec{w}$ and b. To determine $\vec{w}$ and b, some constraints are required.

Using Figure 2 as an example, to determine $\vec{w}$ and b's values, for rectangular vectors which is farther than the center-line, find two variables $\vec{w}$ and b that satisfy the equation $\vec{w} \cdot \vec{x_{rec}} + b \geq 1$. The output will be greater than 1 for all rectangle vectors. In addition, the previously found $\vec{w}$ and b should satisfy the equation $\vec{w} \cdot \vec{x_{tri}} + b \leq -1$ for triangle vectors.

However, finding the values of $\vec{w}$ and b using these two equations is not an easy task. For mathematical convenience, let's define a variable y_i which returns a value of +1 for rectangle vectors and -1 for triangle vectors. Therefore, for the equation of the rectangle, $\vec{w} \cdot \vec{x_{rec}} + b \geq 1$ the expression involving

y_i: $y_i \cdot (\vec{w} \cdot \vec{x_{rec}} + b) \geq 1$ and for the triangle: $y_i \cdot (\vec{w} \cdot \vec{x_{tri}} + b) \geq 1$. They both have the same equations $y_i \cdot (\vec{w} \cdot x_{rec\ \vec{or}\ tri} + b) - 1 \geq 0$. In a special case, for the vectors inside the widest street area (gutter) the equation holds $y_i \cdot (\vec{w} \cdot x_{rec\ \vec{or}\ tri} + b) - 1 = 0$.

3.3 Get the width of the street



6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
$\frac{2}{\|W\|}$
$\vec{x}_{\blacksquare}$
$\vec{x}_{\blacksquare} - \vec{x}_{\blacktriangle}$
$\vec{x}_{\blacktriangle}$
$\vec{w}$

Figure 3: Width of the street

What is the widest path to divide triangles and rectangles? Despite not knowing the path widths, we can calculate the difference between the two support vectors (the closest two vectors to the gutter) of the triangles and rectangles. Using these two support vectors as $\vec{x_{rec}}$ and $\vec{x_{tri}}$, respectively, we can calculate the difference between the two vectors as $(\vec{x_{rec}} - \vec{x_{tri}})$. We can determine the width of the street by taking the dot product of the difference between these two vectors and the unit normal to the center-line. In other words, if we have a unit vector pointing in the direction of the center-line, we can calculate the street width.

$$width \ = \ (\vec{x_{rec}} - \vec{x_{tri}}) \cdot \frac{\vec{w}}{\|w\|}$$

From the previous margin constraints, we have the equation, $y_i \cdot (\vec{w} \cdot x_{rec\ \vec{or}\ tri} + b) - 1 = 0$. If we have the rectangle support vector, y_{rec} will be 1 and rest of the equation will be $(\vec{w} \cdot \vec{x_{rec}} + b) - 1 = 0$. So $\vec{x_{rec}} \cdot \vec{w}$ will be $1 - b$. Similarly, when we have the triangle support vector, y_{tri} will be -1 and $\vec{x_{tri}} \cdot \vec{w}$ will be $-1 - b$. With this equations we got $\vec{x_{rec}} = 1 - b$ and $\vec{x_{tri}} = -1 - b$. So the width is $(\vec{x_{rec}} - \vec{x_{tri}}) \cdot \frac{\vec{w}}{\|w\|} = \frac{\vec{x_{rec}}\vec{w} - \vec{x_{tri}}\vec{w}}{\|w\|} = \frac{1-b-(-1-b)}{\|w\|} = \frac{2}{\|w\|}$.

We got the width: $\frac{2}{\|w\|}$. We want to maximize the width. For mathematically convenience, it is okay to maximize $\frac{1}{\|w\|}$. Which also leads to minimize $\|w\|$. Eventually it is minimizing $\frac{1}{2} \cdot \|w\|^2$.

3.4 Maximize Width

From previous section, we learn in order to maximize the width of the street, need to minimize $\frac{1}{2} \cdot \|w\|^2$. To find the min $\|w\|$ we use Lagrange multiplier.

Applying Lagrange Multiplier,

$$L = \frac{1}{2} \|w\|^2 - \sum \alpha_i [y_i (\vec{w} \cdot \vec{x_i} + b) - 1]$$

In order to find the minimum, derivative the equation L and find w, b which makes 0.

$$\frac{dL}{dw} = \vec{w} - \sum \alpha_i y_i \vec{x_i} => 0$$

$$=> \vec{w} = \sum \alpha_i y_i \vec{x_i}$$

$\vec{w}$ is the linear sum of these vectors α_i, y_i, $\vec{x_i}$

$$\frac{dL}{db} = -\sum \alpha_i y_i => 0$$

Plug back in founded derivative to the original Lagrange Multiplier.

$$L = \frac{1}{2} \sum (\alpha_i y_i \vec{x_i}) \cdot \sum (\alpha_j y_j \vec{x_j}) - \sum (\alpha_i y_i \vec{x_i}) \sum (\alpha_j y_j \vec{x_j}) - \sum \alpha_i y_i b + \sum \alpha_i$$

we know b is constant so takes it outside of sum.

$$L = \frac{1}{2} \sum (\alpha_i y_i \vec{x_i}) \cdot \sum (\alpha_j y_j \vec{x_j}) - \sum (\alpha_i y_i \vec{x_i}) \sum (\alpha_j y_j \vec{x_j}) - b \sum \alpha_i y_i + \sum \alpha_i$$

From derivative of b, $-\sum \alpha_i y_i = 0$.

$$L = \frac{1}{2} \sum (\alpha_i y_i \vec{x_i}) \cdot \sum (\alpha_j y_j \vec{x_j}) - \sum (\alpha_i y_i \vec{x_i}) \sum (\alpha_j y_j \vec{x_j}) + \sum \alpha_i$$

$$= \sum \alpha_i - \frac{1}{2} \sum (\alpha_i y_i \vec{x_i}) \cdot \sum (\alpha_j y_j \vec{x_j})$$

$$= \sum \alpha_i - \frac{1}{2} \sum \sum a_i a_j y_i y_j \cdot x_i \cdot x_j$$

We see that the optimization is only depends on the pair of samples.

3.5 Apply back to Decision Rule

Now plug w back in decision rule from the figure 1.

$$\sum (a_i y_i \vec{x_i}) \cdot \vec{u} + b \geq 0, \ then \ Rectangle$$

4 Kernel Trick

In SVM, sometimes the data isn't linearly separable. In this case, a non-linear transformation must be applied to the data so that it can then be categorized. Then, SVM doesn't need to know the function for transformation, it just needs to know how the data points compare to each other after they're all transformed by using the kernel function which is centered around the dot product.

In the linear kernel, $f(x) = x$ and the kernel function is $k(x, x') = x^T x'$. This attempts to create a linear boundary between data points with minimal transformation, so it often isn't enough to create a proper boundary.

In the polynomial kernel, the equation of the transformation would be $f(x) = (x_1, x_2, x_1 x_2, x_1^2, x_2^2)$ and the polynomial kernel is $k(x, x') = (1 + x^T x')^2$. In this case, it is particularly useful because you don't need to know the transformation function itself. This will create a boundary that appears to be much more dynamic because the transformation that is applied is non-linear.

In the RBF (Radial Basis Function) kernel, the transformation function captures infinite dimensions so the abstraction of $f(x)$ in the kernel function is necessary to properly fit the data. Instead we just use the simplified kernel function: $k(x, x^{'}) = e^{-r \left\| x - x^{'} \right\|^2}$.

5 Experiments

5.1 PCA Result

Based on our research on PCA and SVM methods, we applied Support Vector Regression (SVR) to our housing price dataset, aiming to predict house prices using 80 features. The dataset contains a large number of features, so we used PCA to reduce the number of feature columns to 8.

To apply PCA, we first scaled our data so that the eigendecomposition wasn't skewed and then we used sklearn's PCA method to reduce the dimensionality of our dataset to 8 columns by projecting the data onto the space spanned by these 8 eigenvectors. We then centered these results around the origin by transforming the data.

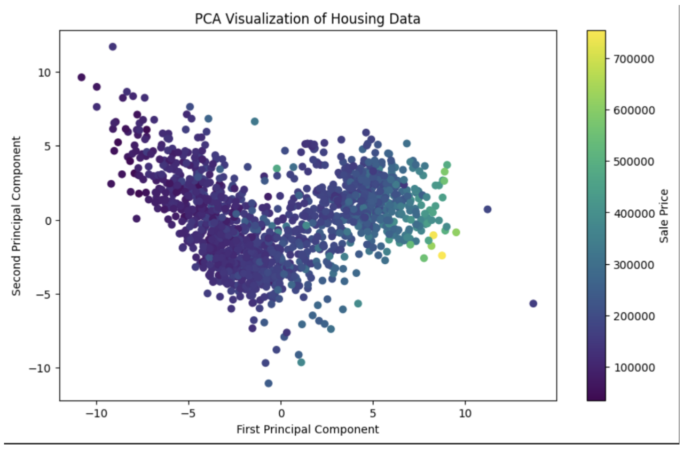


Figure 4: PCA Plot of YearBuilt and Foundation_CBlock

These 8 features that captured the greatest variance were, in order from greatest magnitude to least: YearBuilt, Foundation_CBlock, GrLivArea, 2ndFlrSF, MSSubClass, BldgType_1Fam, BldgType_Duplex, and SaleCondition_Normal.

One thing to keep in mind is that the principal components that we choose to retain are those that capture the greatest variance, not necessarily those that have the greatest impact on the dataset. These components are not directly interpretable in terms of the original features but represent patterns of variation in the data. Thus, we can draw very limited conclusions from the results of the PCA.

As mentioned in the resources section above, when we tried to run the Support Vector Regression model on the original data, the runtime disconnected after roughly 1.5 hours rather than completing the computation. This means the computation was incredibly demanding and had to be abandoned. This demonstrates the value of PCA in increasing efficiency of training a model. Without PCA, we couldn't even train the model beause it was too inefficient due to the size of the data, but in the reduced form, it took well under a minute to fit the model for the PCA adjusted data.

5.2 Support Vector Regression Model

The experiment dataset underwent PCA, and the eight columns with the highest variance were selected. These datasets were then used to split the data into a training set and a test set in a 7:3 ratio. This ensured that we could accurately train our data, but wouldn't overfit so testing would still be accurate. The models were trained on this split data, and accuracy was evaluated.

We conducted experiments using three models: linear, RBF, and polynomial, based on the kernel trick used before calculating the support vector.

5.2.1 Fine Tuning

SVR models, contain several parameters, but I will only mention the parameters we touched.

- kernel : {'linear', 'rbf', 'poly'} The types of kernel trick we will use in our SVR Model
- C : { float, default = 1.0 (Positive) } Regularization parameter.

- degree : { default = 3 (Non-Negative) } Degree of the polynomial kernel function.
- gamma : { 'auto', 'scale' or float default = 'scale'} Kernel Coefficient for RBF, Poly, and Sigmoid SVR
- coef0 : { default = 0.0 (Non-Negative float) } Independent term in kernel function. Only applicable for Poly and Sigmoid

We used GridSearchCV to calculate the optimal parameters for 3 models, which search over specified arrays of parameter values. In order to see the significant difference in the model score, we set the input parameter array as follows.

- kernel : {'linear', 'rbf', 'poly' }
- C : [0.1, 1, 10, 100, 1000]
- degree : { 3, 5, 8}
- gamma : { 'auto', 'scale', 0.1, 0.01, 0.001, 0.0001}
- coef0 : { 10, 25, 50, 100}

5.3 Result

We applied GridSearchCV through three different kernels to SVR: the Linear Kernel, the Poly Kernel, and the RBF Kernel. Based on the results of the experiments, the R-squared scores are as follows:

Grid Search Parameter result			
Parameters	Linear Kernel Model	RBF Kernel Model	Poly Kernel Model
Kernel	linear	rbf	poly
C	1350	1000	210
degree	1	Not-applicable	3
gamma	Not-applicable	0.01	scale
coef0	Not-applicable	Not-applicable	25

Linear SVR R-squared score was: 0.7769931267399243
RBF SVR R-squared score was: 0.5085046369569125
Polt SVR R-squared score was: 0.8689104943523224

As observed, using Poly kernel generated better results. We used the R-squared score as the scoring function. $R^2 = 1 - \frac{u}{v}$, where $u = \sum(y_{true} - y_{predict})^2$ and $v = \sum(y_{true} - y_{true\ mean})^2$. A value closer to 1 indicates a better model fit.

5.3.1 Hypothesis to the result:

Based on comprehensive analysis, the Poly kernel performed better than the other two kernels, RBF and Linear kernel applied models. According to our hypothesis, polynomial kernel has better estimation than others due to the trend that more features lead to higher house prices, but no clear separation exists between the data. Separating the datasets with a liner kernel would not be enough, and RBF has a too complex way of setting support vectors. So the polynomial has some terms to distinguish the dataset, but they are not too complex like RBF.

Regularization strength is determined by the inverse proportion of C values. The C values are very high after parameter tuning. In our assumption, there is no clear separation in the dataset, so less regularization helps models perform better. If there was a lot more variance between data, regularization would help us generalize the model and counteract overfitting.

The coef0 values are only applicable to Polynomial kernel. Considering that it determines the number of independent terms in kernel functions, the polynomial model had a better result.

There is one aspect of the experiment that is unexpected: the testing score is higher than the training score. Here is the table of training and testing accuracy. (In the method of calculating scores, we used the R2 Score function).

Calculate training score	
Linear SVR R-Squared Score	0.7688902758375236
RBF SVR R-Squared Score	0.5168741555025075
POLY SVR R-Squared Score	0.8697297388733204

Calculate Testing Score	
Linear SVR R-Squared Score	0.8548107563261176
RBF SVR R-Squared Score	0.5498264522835576
POLY SVR R-Squared Score	0.8937439503876586

Based on both training and testing accuracy results, we found that the testing scores were slightly higher than the training scores. Using the SVR models, the testing prices of houses are predicted based on the training inputs. Our assumption was that the results might be misleading. In order to ensure that the result is correct, we trained and tested a linear regression model with the same PCA applied dataset.

The linear regression result is as follows.

Linear Regression model results	
Linear regression training score	0.9295112045275672
Linear regression testing score	0.7599034228795364

Using linear regression, it clearly shows that the training score is about 17% higher than the testing score. Consequently, the results would suggest that the interpretation is more likely to be accurate. Additionally, we can assume that the unexpectedly high accuracy of the testing results of the POLY SVR can be attributed to very fortunate tuning results, so in this case the POLY SVR model is still valid and the most accurate.

6 Conclusion

One of our greatest challenges throughout this process was managing the sheer magnitude of this data, especially due to the frequency of categorical variables. The original dataset had 80 features, but after one-hot encoding all of the categorical variables and preprocessing, it grew to 271 columns. This limited the amount of direct testing we could do due to runtime issues, especially as we did more intensive fitting for our SVM model. In the future, we would like to expand this project by trying some different models such as XG Boost. Since it is a gradient boosting tree model, based on each feature it makes separate trees and combines it all at once so it is better than the basic tree method and could provide better insights on the categorical data than what all of our one-hot encoding can do. Alternatively, we would like to explore other options outside of one-hot encoding for capturing the categorical variables. I believe that better preprocessing could greatly improve the accuracy of our results.

References

[1] Stanley H Chan. “Introduction to probability for data science”. In: *(No Title)* (2021).

[2] Mark Crovella. *Applications of the SVD - Linear Algebra, Geometry, and Computation*. Accessed 21 Apr. 2024. n.d. URL: https://mcrovella.github.io/CS132-Geometric-Algorithms/L26ApplicationsOfSVD.html.

[3] IntuitiveMachineLearning. *Support Vector Machines: All You Need to Know!* Accessed 23 Apr. 2024. 2020. URL: https://www.youtube.com/watch?v=ny1iZ5A8ilA&ab_channel=IntuitiveMachineLearning.

[4] MIT OpenCourseWare. *16. Learning: Support Vector Machines*. Accessed 21 Apr. 2024. 2014. URL: https://www.youtube.com/watch?v=_PwhiWxHK8o&ab_channel=MITOpenCourseWare.

[5] *Regularization*. URL: https://www.ibm.com/topics/regularization.

[6] *SciKitLearn Documentation*. Accessed 21 Apr. 2024. URL: https://scikit-learn.org/stable/index.html.

[7] *SciKitLearn Support Vector Regression Documentation*. Accessed 23 Apr. 2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html.

[8] VisuallyExplained. *The Kernel Trick in Support Vector Machine (SVM)*. Accessed 30 Apr. 2024. 2022. URL: https://www.youtube.com/watch?v=Q7vT0--5VII&ab_channel=VisuallyExplained.