Final Project Write-up

Group member: Rachel Chen (rc4889), Yuzuka Rao (tr2371)

April 2025

1. Project Overview

The topic of our project is to predict the house price range of a California house based on some basic information about the house. The dataset we use is a dataset that contains 20641 pieces of information about houses, each containing features: longitude, latitude, housing median age, total rooms, total bedrooms, population, households, median income, median house value and ocean proximity. The median_house_value is the target variable we want to predict. The types of features are shown below:

longitude	float64
latitude	float64
housing_median_age	float64
total_rooms	float64
total_bedrooms	float64
population	float64
households	float64
median_income	float64
median_house_value	float64
ocean_proximity	object

We grouped these housing prices into 17 categories, each spanning $30,000, with the goal of predicting which category a median_house_value would fall into. For example, one category can be from 200 thousand dollars to 230 thousand dollars.

2. Unsupervised Analysis

2.1. Data cleaning & processing

When doing data cleaning, we find that the dataset overall is clean so it doesn't require much work on data cleaning. The dataset has some null values in column "total_bedrooms" which are cleaned by running the code below:

```
[5]: median_value = df['total_bedrooms'].median()
     df['total_bedrooms'] = df['total_bedrooms'].fillna(median_value)

[6]: df.isnull().sum()
```

[6]:

	0
longitude	0
latitude	0
housing_median_age	0
total_rooms	0
total_bedrooms	0
population	0
households	0
median_income	0
median_house_value	0
ocean_proximity	0

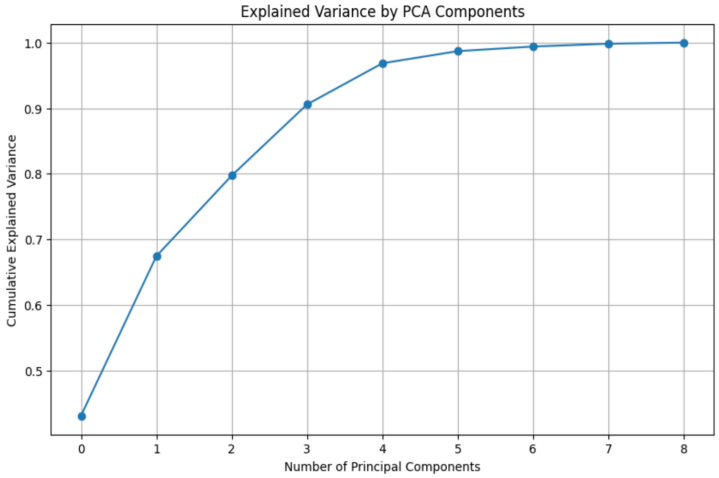
Originally, the "ocean_proximity" data are not integers so we do ordinal encoding to convert strings to numeric values. With a higher value, the house is closer to the ocean.

```
# Step 4: Encode 'ocean_proximity'
ocean_proximity_mapping = {
    '<1H OCEAN': 3,
    'NEAR OCEAN': 2,
    'NEAR BAY': 1,
    'INLAND': 0
}
housing['ocean_proximity'] = housing['ocean_proximity'].map(ocean_proximity_mapping)
housing = housing.dropna()
```

Then, we categorize median_house_value into 17 ranges spanning $30,000 each. Since we are doing a classification problem, we need to convert the target "median_house_value" into categorical numerical values from 0 to 16. Thus, we replace the original target median_house_value with price_range. The value of price_range is between 0 to 16 inclusively, and price_range = 0 means the median_house_value falls between $0 to $30,000.

2.2 PCA and clustering

We first use PCA to help us reduce data collinearity and complexity. After separating the dataset into features X and target Y, we implement PCA to get the top 5 principal components.

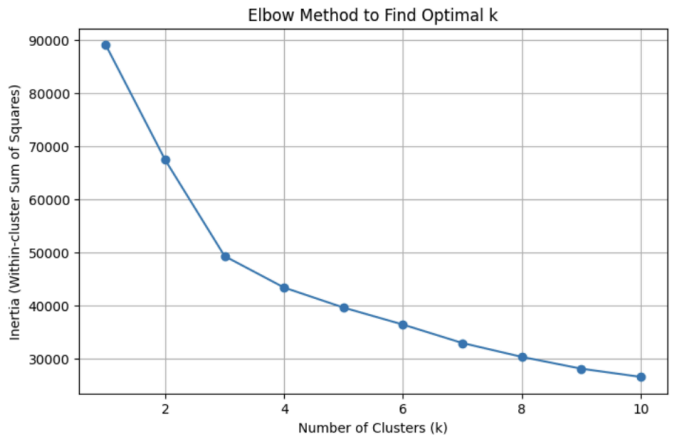



```
]: X_pca_5 = X_pca[:, :5]
print("Shape of reduced X_pca_5:", X_pca_5.shape)
cumulative_variance_5 = np.sum(pca.explained_variance_ratio_[:5])
print(f"Cumulative explained variance of first 5 principal components: {cumulative_variance_5:.4f} ({cumulative_variance_5*100:.2f}%)")

Shape of reduced X_pca_5: (10214, 5)
Cumulative explained variance of first 5 principal components: 0.9684 (96.84%)
```

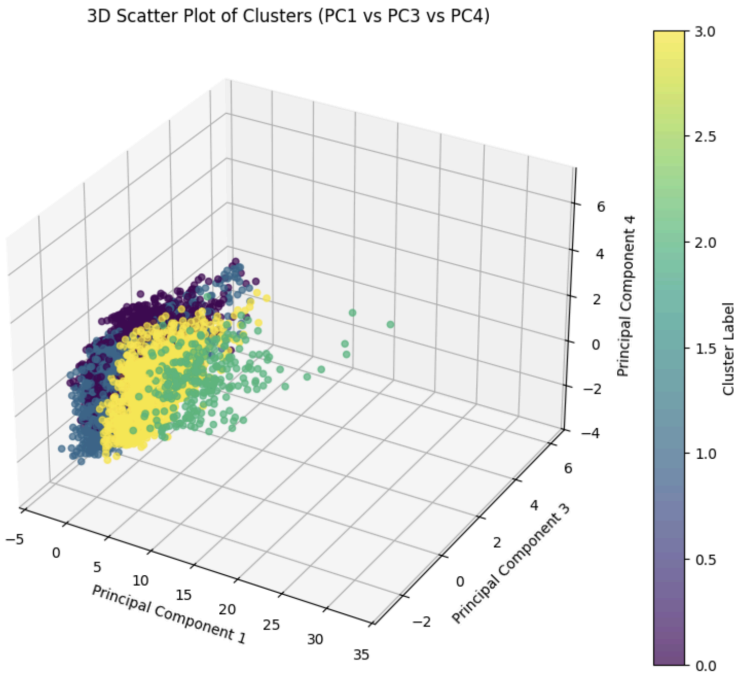
By implementing 5 components, we retain 96.84% of the original data's variance, effectively reducing dimensionality while preserving most of the information.

After getting these components, we implement the Elbow Method to plot the inertia against the number of clusters graphs to help us decide how many clusters we can use.



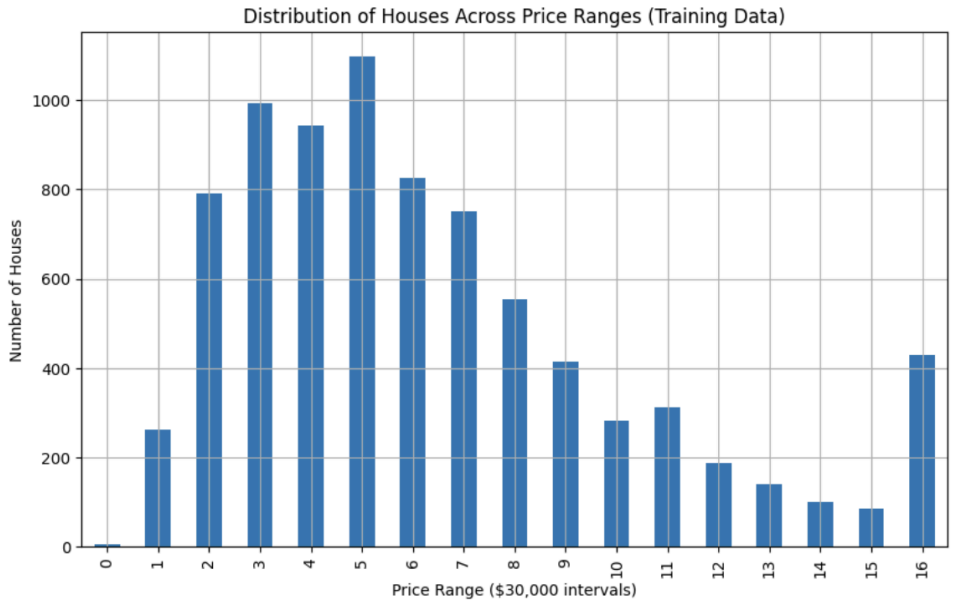
We find that when we have 4 clusters, the inertia is low and adding more clusters will not have strong improvement on the model. The data points are close enough to the center of each cluster. So we calculate the distance between each data point and each cluster, and adding these 4 new features as distance_to_cluster_n (n from 0 to 3) into our dataset. We apply K Means clustering algorithm to get the clusters. Cluster 0 to 3 each has 3952, 4340, 261 and 1661 samples.

The figure below is a visualization of our clustering. Since there are more than 3 principal components, they cannot be shown together. We choose 3 components as examples to visualize.

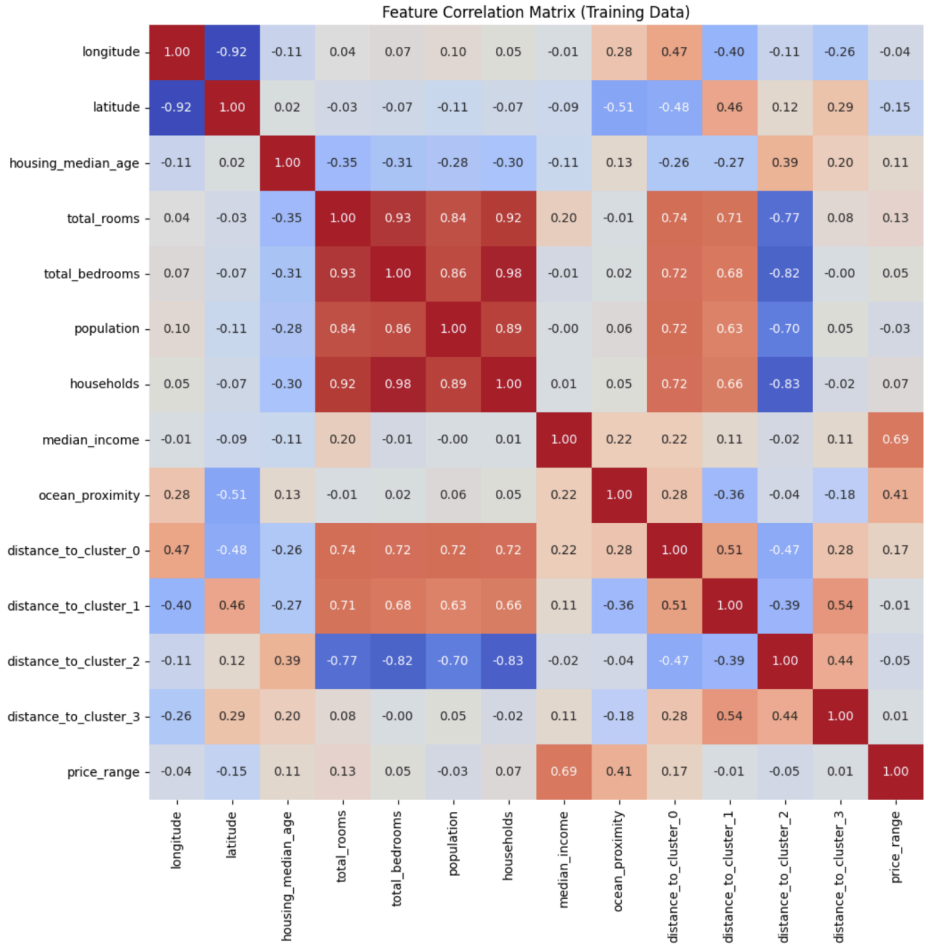


2.3 Data Visualization

After splitting the dataset into training set and testing set, we plot the median_house_value distribution:



Since the data appears to be a left-skewed distribution without a highly unbalanced distribution, we choose not to use a data balancing method like SMOTE. However, the correlation matrix shows that several features have a high correlation with each other:



For example, households and total_bedrooms have high correlation with each other, so their high collinearity can apply a high bias to the model training. The Kmeans clustering and the new features of the distance between each housing data and each cluster center can help the models reduce bias caused by collinearity.

We also create the scatter plot of the relationship between price range and each feature:

These plots could help visualize whether a feature has strong impact on the price_range. For example, when price_range get larger, longitude and latitude becomes more concentrated in a smaller range, which could mean that the housing price in a specific area could be higher than those in other areas. The other example is that when price_range becomes larger, median_income becomes larger. This could mean that houses with higher price has a relatively strong relationship with people with higher income. The examples of weak relationships, like distance to clusters and housing median age, shows that when price_range becomes larger, the values of such features does not have a significant change.

In conclusion, after data cleaning and processing, our dataset now contains features longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, ocean_proximity, and 4 distances to each cluster, and our target is predicting price_range.

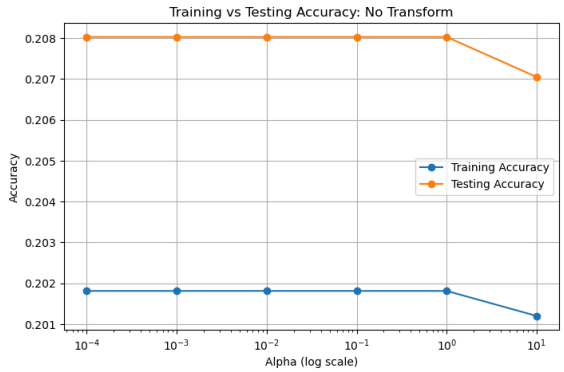
3. Supervised Analysis

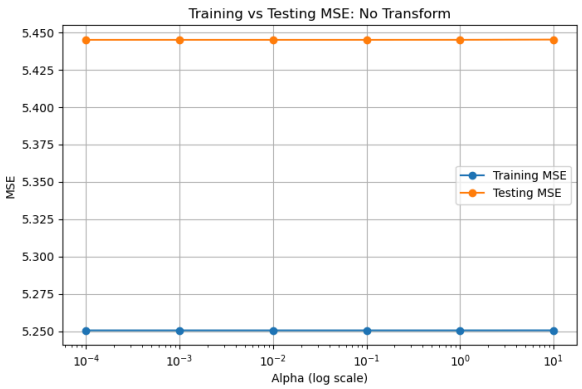
3.1 Linear Regression

Since the original dataset represents house values as continuous numbers, our prediction task began from a regression perspective. We first attempted to use linear regression to directly predict price ranges by treating them as continuous values and rounding the outputs to the nearest integer. However, this approach produced poor results. We applied a simple linear model without any feature transformation or regularization and the training accuracy hovered around 20.18%, with the testing accuracy slightly higher at about 20.8%. The precision and recall scores were similarly low, and both the training and testing mean squared errors (MSE) were high, indicating a strong underfit.

3.1.1 Ridge Regression

We applied Ridge Regression to the original feature set (no transformations). However, adding ridge regularization with different alpha values from 0.001 to 100 only slightly changed the outcome, suggesting that the original feature set lacked the complexity needed for accurate prediction, and the model's capacity was fundamentally limited.

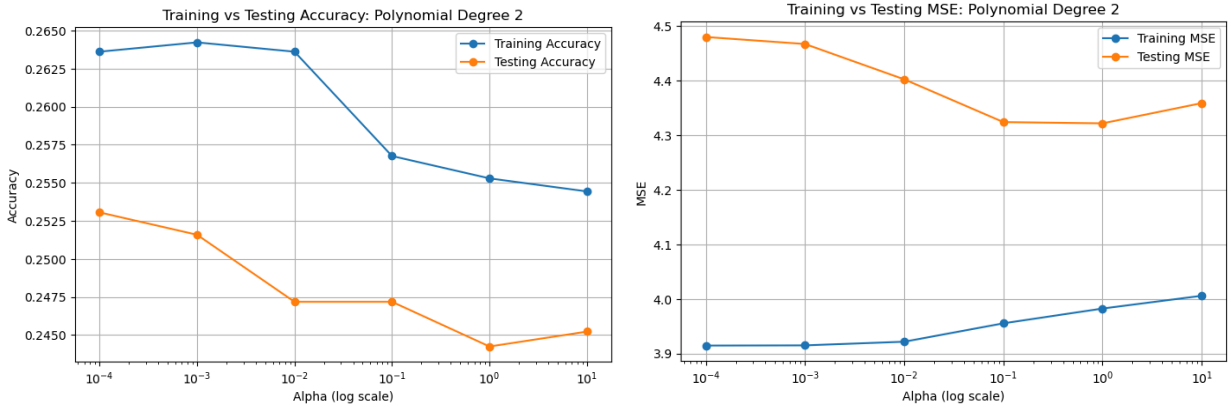




Low α values did not improve performance compared to the baseline (Train ~20.18%, Test ~20.80%). As α increased, performance slightly worsened, suggesting that the model is underfitting even more as regularization becomes stronger.

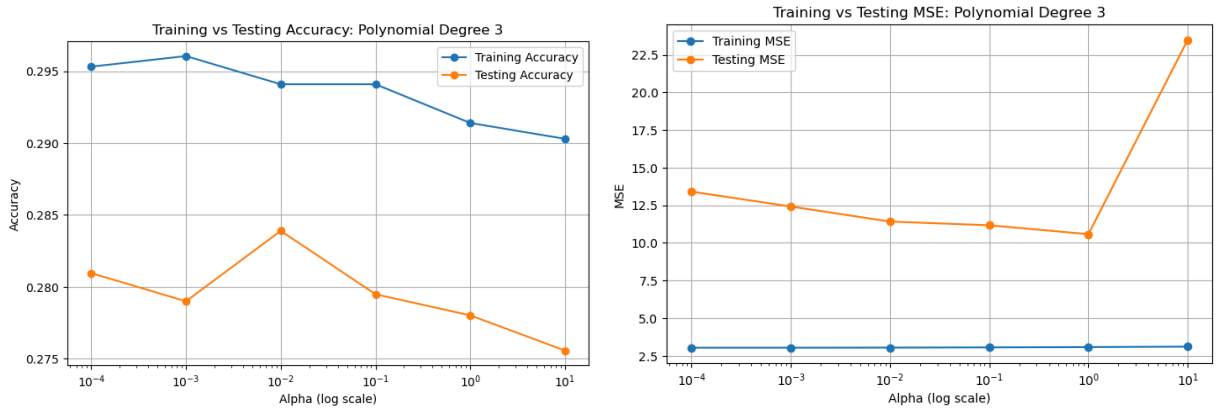
3.1.2 Polynomial Features of degree 2

To introduce more flexibility, we expanded the feature set using a polynomial transformation of degree 2. This improved the training accuracy to around 26.4% and testing accuracy to approximately 25.3% at the best alpha values (around 0.0001–0.001). However, we find that as the regularization strength increased, both training and testing performance declined slightly. While polynomial degree 2 helped the model capture some interactions between features, it also introduced mild overfitting, which regularization partially controlled. Overall, polynomial degree 2 transformation provided a modest improvement compared to the untransformed model. It balances additional model complexity with acceptable generalization.



3.1.3 Polynomial Features of degree 3

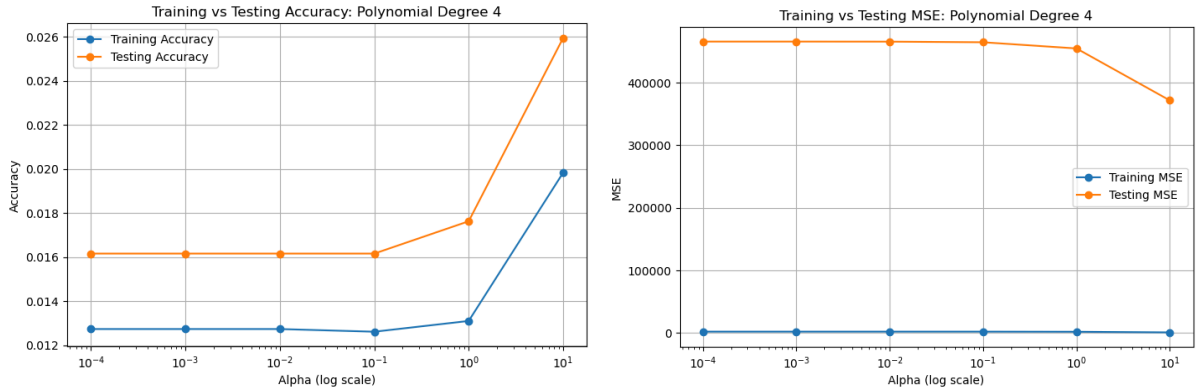
Further increasing the polynomial degree to 3 introduced more complex interactions into the feature space. With this transformation, we get a higher training accuracy of about 29.5%, and the test accuracy peaked around 28.4%. However, the model started to exhibit stronger overfitting tendencies: the gap between training and testing performance widened, especially at low regularization values. We find it interesitng that while regularization helped mitigate some overfitting, even at optimal alpha values, the model failed to significantly outperform the degree 2 polynomial case on the test set. This suggests that the added complexity provided diminishing returns for generalization.



Even though this shows an overfit, we still try degree 4 to see if it can give a better accuracy.

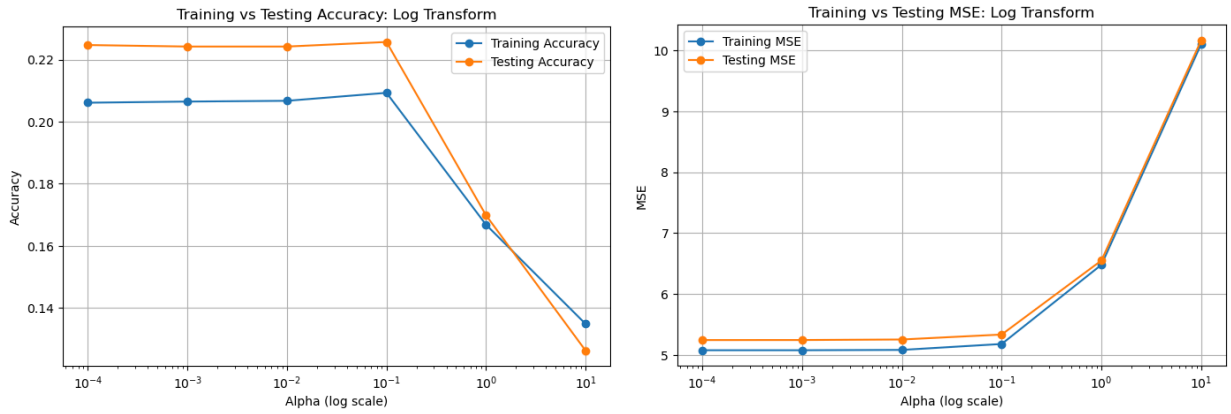
3.1.4 Polynomial Features of degree 4

With a polynomial degree of 4, the model's performance sharply deteriorated. Although the training accuracy initially rose slightly at very small alpha values, it remained extremely low overall (~1.2%–2%) compared to previous transformations. Testing accuracy was similarly poor (~1.6%–2.6%), and the MSE skyrocketed to extremely high values, indicating catastrophic overfitting or numerical instability. We find that regularization with higher alpha values cannot rescue the model. Thus, a polynomial degree of 4 was far too complex for the available data and introduced excessive noise sensitivity.



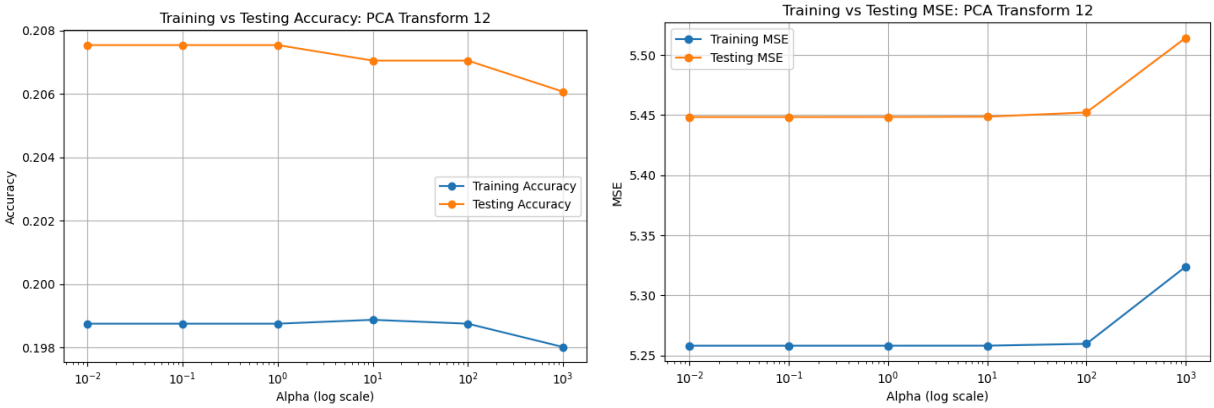
3.1.5 Log Transform

In addition to polynomial expansions, we applied a log transformation to the features to reduce skewness and stabilize variance. The log-transformed model achieved a training accuracy around 20.6% and a slightly higher testing accuracy (~22.5%), with the best results at small alpha values. Although the improvements over no transformation were moderate, the log transformation helped the model generalize slightly better without introducing significant overfitting. However, when regularization was increased aggressively (alpha > 1), model performance deteriorated, indicating that heavy penalties made the model too simple to capture even basic patterns.



3.1.6 PCA Transformation

Finally, we applied Principal Component Analysis (PCA) to reduce feature dimensionality while preserving most of the data's variance. Using 12 principal components, the PCA-transformed model showed very similar performance to the original untransformed model, with training and testing accuracies both around 20.7%. Across all alpha values tested, performance remained stable. This can show that PCA compressed the information without significantly altering the model's predictive capacity. However, the reduced complexity did not provide any strong gains either, suggesting that the key limitation was not dimensionality but the model's linear nature.



3.1.7 Conclusion and Table

Overall, the best performance from linear regression models was achieved by applying a polynomial degree 3 transformation with light regularization. This approach provided a moderate balance between complexity and generalization. However, the limited accuracy and signs of underfitting across most transformations suggested that linear models were insufficient for the complexity of the house price prediction task, motivating the exploration of nonlinear models like Support Vector Machines.

Transformation	Alpha	Train Accuracy	Test Accuracy	Precision	Recall	Train MSE	Test MSE
No Transform	0.001	0.20181128	0.20802741	0.21924137	0.20802741	5.25050684	5.44496052
No Transform	0.01	0.20181128	0.20802741	0.21924137	0.20802741	5.25050684	5.44496067
No Transform	0.1	0.20181128	0.20802741	0.21924137	0.20802741	5.25050684	5.44496212
No Transform	1	0.20181128	0.20802741	0.21924137	0.20802741	5.25050706	5.4449768
No Transform	10	0.20119936	0.20704846	0.21857622	0.20704846	5.25052879	5.44514074
No Transform	100	0.19960837	0.20606951	0.21750721	0.20606951	5.25235748	5.44814359
Polynomial Degree 2	0.0001	0.26361523	0.25305923	0.25917804	0.25305923	3.91462919	4.48040164
Polynomial Degree 2	0.001	0.26422715	0.2515908	0.25783602	0.2515908	3.91495905	4.46752367
Polynomial Degree 2	0.01	0.26361523	0.24718551	0.2536642	0.24718551	3.92164787	4.40274285
Polynomial Degree 2	0.1	0.25676172	0.24718551	0.2531849	0.24718551	3.95535975	4.3243544
Polynomial Degree 2	1	0.25529311	0.24424865	0.2506027	0.24424865	3.98247838	4.32204434

Polynomial Degree 2	10	0.25443642	0.24522761	0.25074188	0.24522761	4.00588941	4.35914484
Polynomial Degree 3	0.0001	0.29531269	0.28095937	0.28533232	0.28095937	3.03635971	13.4056103
Polynomial Degree 3	0.001	0.296047	0.27900147	0.28341366	0.27900147	3.03688763	12.4151995
Polynomial Degree 3	0.01	0.29408885	0.28389623	0.28806956	0.28389623	3.04210846	11.4171152
Polynomial Degree 3	0.1	0.29408885	0.27949095	0.28342801	0.27949095	3.05789533	11.1696568
Polynomial Degree 3	1	0.2913964	0.27802252	0.28250267	0.27802252	3.07744331	10.5800908
Polynomial Degree 3	10	0.29029495	0.27557514	0.27938197	0.27557514	3.10782651	23.4866304
Log Transform	0.0001	0.20609473	0.2246696	0.23492577	0.2246696	5.07885931	5.24593479
Log Transform	0.001	0.20646188	0.22418013	0.23438296	0.22418013	5.07893929	5.24654135
Log Transform	0.01	0.20670665	0.22418013	0.2349833	0.22418013	5.08423641	5.25498164
Log Transform	0.1	0.20927671	0.22564856	0.23769644	0.22564856	5.18137074	5.3364439
Log Transform	1	0.16680945	0.16984826	0.18165173	0.16984826	6.48172266	6.55195939
Log Transform	10	0.13486721	0.12628488	0.10491607	0.12628488	10.105538	10.1668913
Polynomial Degree 4	0.0001	0.01272794	0.01615272	0.09770767	0.01615272	2278.58397	465806.298
Polynomial Degree 4	0.001	0.01272794	0.01615272	0.09770767	0.01615272	2278.37282	465796.215
Polynomial Degree 4	0.01	0.01272794	0.01615272	0.09770767	0.01615272	2276.26297	465695.398
Polynomial Degree 4	0.1	0.01260556	0.01615272	0.09771431	0.01615272	2255.33011	464689.255
Polynomial Degree 4	1	0.01309509	0.01762115	0.1067968	0.01762115	2061.44886	454824.298
Polynomial Degree 4	10	0.01982622	0.02594224	0.10911156	0.02594224	1009.5648	371790.817
PCA Transform 12	0.01	0.19875168	0.20753793	0.21942522	0.20753793	5.25805211	5.4482268
PCA Transform 12	0.1	0.19875168	0.20753793	0.21942522	0.20753793	5.25805211	5.44822925
PCA Transform 12	1	0.19875168	0.20753793	0.21942522	0.20753793	5.25805229	5.44825394
PCA Transform 12	10	0.19887407	0.20704846	0.21904057	0.20704846	5.25807016	5.44851213
PCA Transform 12	100	0.19875168	0.20704846	0.21898172	0.20704846	5.25963401	5.45204486
PCA Transform 12	1000	0.19801738	0.20606951	0.21332266	0.20606951	5.32375793	5.51404725

3.1.8 Improvement

Since using the classified data doesn't perform well, we revert the data back to continuous raw house value data, and trained a linear regression model to predict that directly. After generating predictions, we converted those continuous outputs into discrete price range classes. This method led to a slight improvement in accuracy and a reduction in error, but the model still failed to generalize well, as reflected in the test performance and the evaluation metrics.

No transformation

```
Train Accuracy: 0.2192
Test Accuracy: 0.2173
Train Precision: 0.2746
Test Precision: 0.2724
Train Recall: 0.2192
Test Recall: 0.2173
```

Polynomial Degree 2

```
Train Accuracy: 0.2822
Test Accuracy: 0.2697
Train Precision: 0.3256
Test Precision: 0.3122
Train Recall: 0.2822
Test Recall: 0.2697
```

Polynomial Degree 3

```
Train Accuracy: 0.3076
Test Accuracy: 0.2834
Train Precision: 0.3441
Test Precision: 0.3194
Train Recall: 0.3076
Test Recall: 0.2834
```

Polynomial Degree 4

```
Train Accuracy: 0.3140
Test Accuracy: 0.2805
Train Precision: 0.3522
Test Precision: 0.3124
Train Recall: 0.3140
Test Recall: 0.2805
```

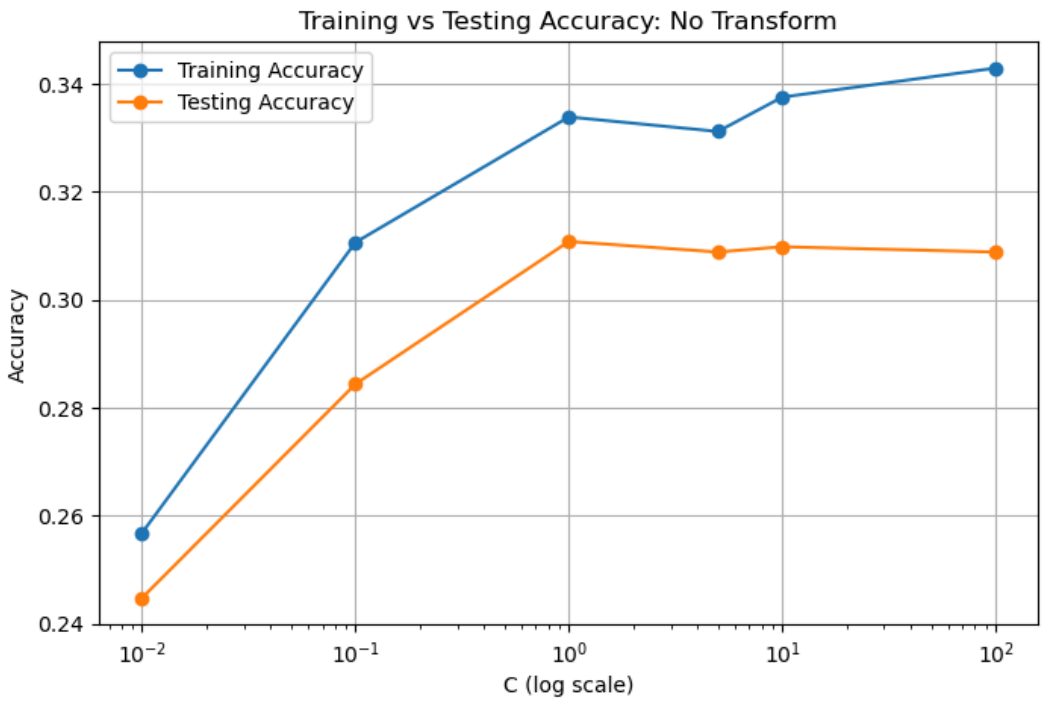
These outcomes revealed that linear regression, even when applied thoughtfully in a regression-to-classification pipeline, was insufficient for capturing the complexities of the price prediction task. Motivating us to use models better fit for classification problems for later training.

3.2 SVM (Support Vector Machine)

The second model we explored for predicting house price ranges was the Support Vector Machine (SVM). We trained several SVM classifiers using different kernel transformations: linear, polynomial (degree 2, 3, 4), and evaluated them across various regularization strengths. Without any feature transformation, the linear SVM achieved moderate performance, with the best test accuracy around 31% at C=1, which already outperformed the baseline linear regression model. As we introduced more complex polynomial kernels, the model captured higher-order interactions between features, resulting in an improvement in training accuracy. However, the use of higher-degree polynomial kernels also caused noticeable overfitting: while training accuracy increased significantly, testing accuracy plateaued or even slightly declined, indicating that the model memorized the training data at the cost of generalization.

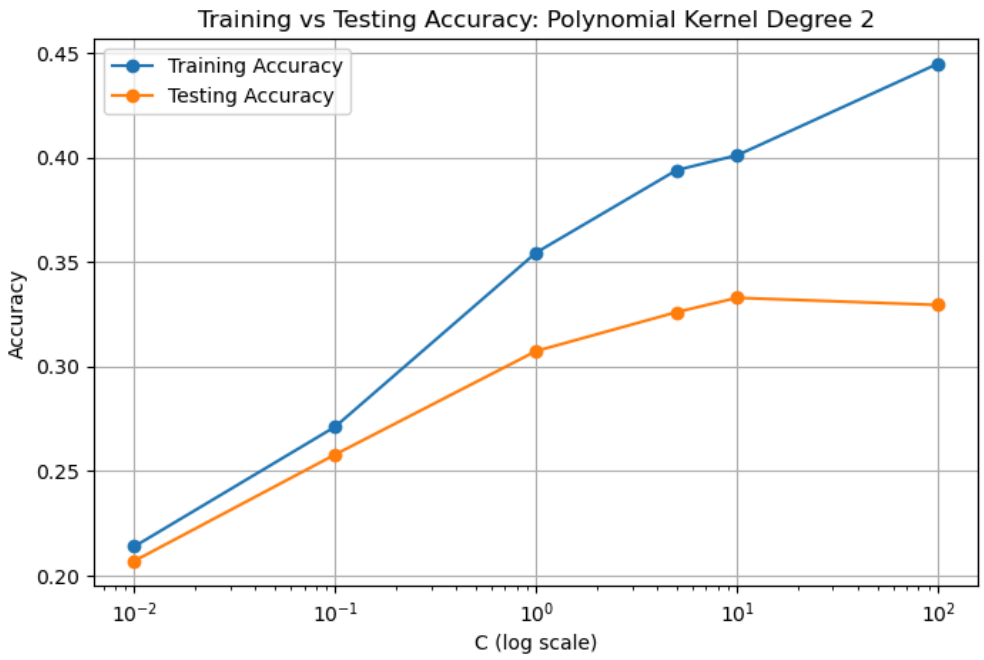
3.2.1 Non-Transformation

We first trained an SVM model using a simple linear kernel without any feature transformations. This model achieved a training accuracy of approximately 33% and a testing accuracy of around 31% at the best C value.



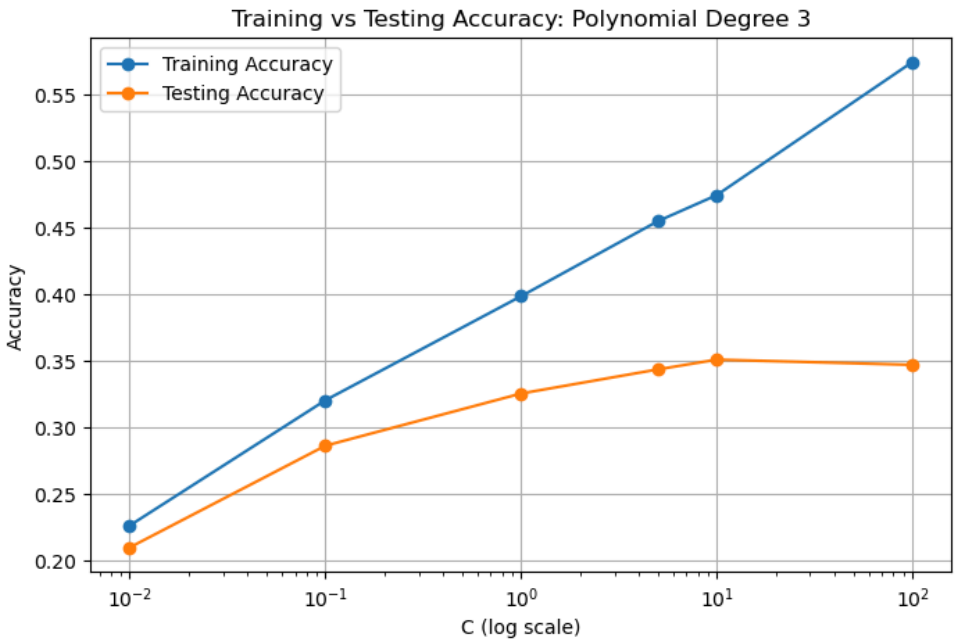
3.2.2 Polynomial Kernel Degree 2

Next, we applied a polynomial kernel of degree 2, which allows the SVM to capture pairwise feature interactions without explicitly creating new features. With degree 2, the training accuracy rose slightly compared to the linear kernel, reaching around 44%, and testing accuracy improved to about 33%. The model benefited from the additional flexibility, although the gains were moderate. Overfitting was present but controlled: the gap between training and testing accuracy was larger than in the linear case but not severe. Regularization with C values around 1–5 provided the best balance between fitting the training data and maintaining generalization.



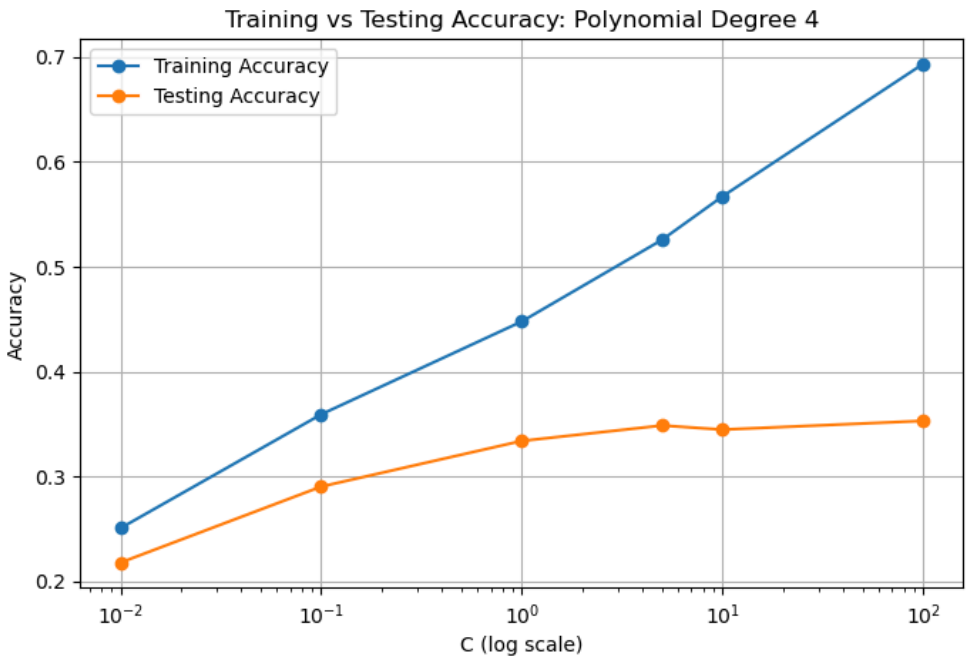
3.2.3 Polynomial Kernel Degree 3

When we increased the kernel degree to 3, the model became more powerful, capturing more complex feature relationships. Training accuracy jumped higher, exceeding 40% at low regularization levels, but testing accuracy did not improve proportionally and remained around 32–33%. Overfitting became much more visible: the model fit the training data increasingly well as C grew, but the testing performance peaked early and then degraded. This showed that while degree 3 polynomials could model the training set's complexity better, the model failed to generalize effectively to unseen data without tight regularization.



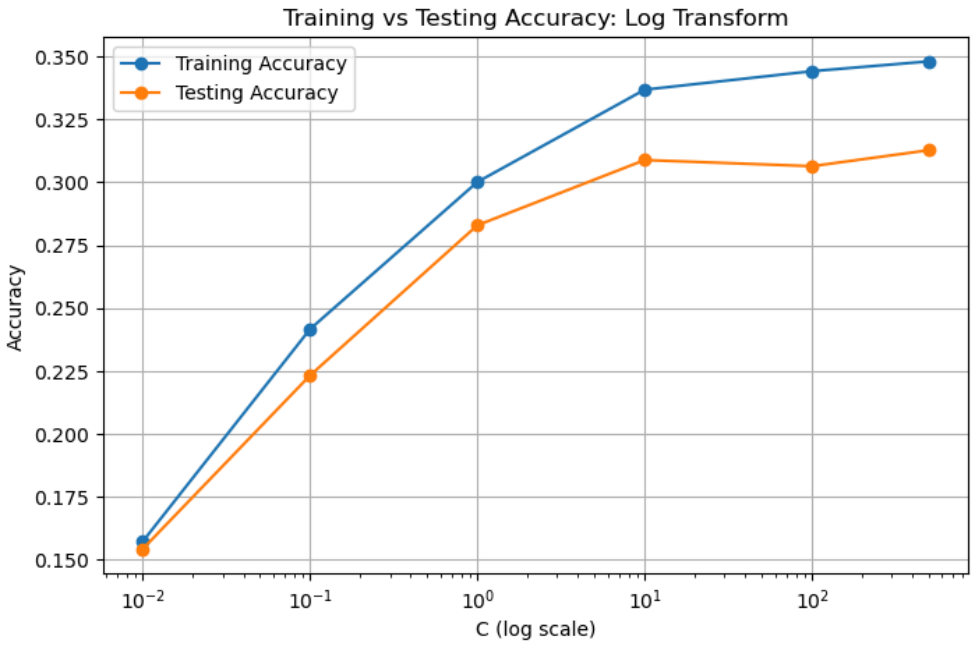
3.2.4 Polynomial Kernel Degree 4

With a polynomial kernel of degree 4, the SVM became highly expressive. Training accuracy reached over 50% at high C values, indicating that the model was fitting intricate patterns in the training data. However, this came at the cost of heavy overfitting: the gap between training and testing accuracy widened significantly, and the testing accuracy plateaued or even slightly decreased compared to degree 3. While degree 4 allowed the model to learn complex structures, it also made the model extremely sensitive to noise and small variations, making it less reliable for generalization. Due to the highly overfitting, we decided to change to another transformation.



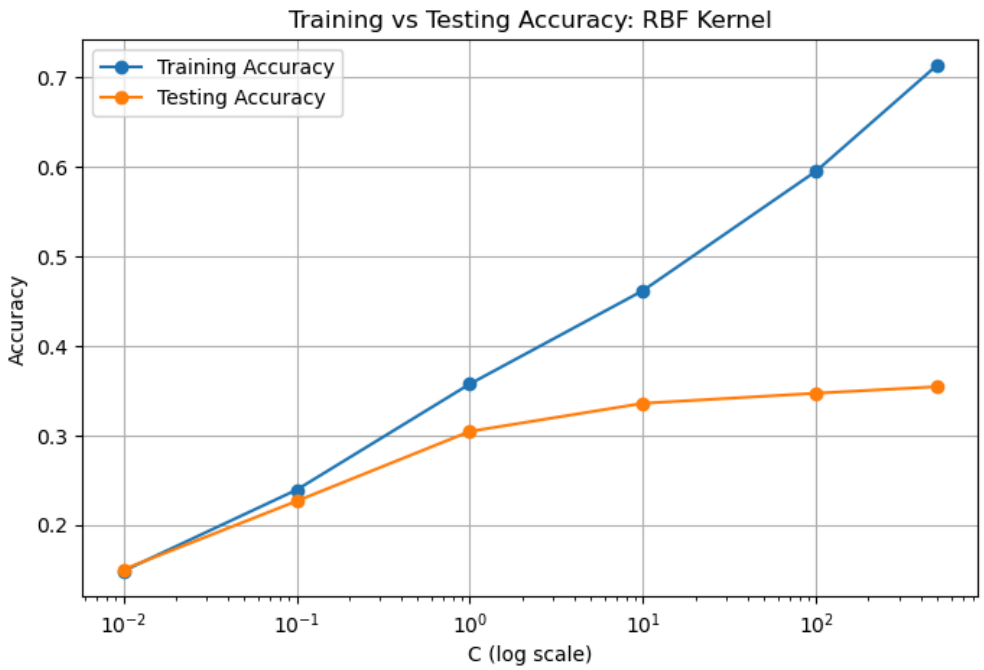
3.2.5 Log Transformation

To address the skewness in the data, we applied a logarithmic transformation to the feature set before training the SVM. Using a linear kernel, the log-transformed features allowed the model to stabilize variance and better distribute the feature values. The training accuracy improved steadily as C increased, reaching around 34.8% at high C values (C=500), while the testing accuracy achieved approximately 31.3%. Compared to the model without transformation, the log transform slightly improved the generalization performance by compressing extreme feature values and reducing the effect of outliers. Although the training accuracy was consistently higher than the testing accuracy, the gap was moderate, indicating that log transformation helped control overfitting while enhancing model expressiveness for moderate C values.



3.2.6 RBF Kernel

In addition to polynomial kernels, we also experimented with the radial basis function (RBF) kernel, which projects the data into an infinite-dimensional feature space. The RBF kernel SVM showed a significant improvement in training accuracy as C increased, reaching over 71% training accuracy at C=500. However, the testing accuracy, while better than linear and polynomial models, was much lower relative to the training accuracy, peaking around 35.4% at high C values. The RBF kernel clearly captured complex nonlinear patterns in the training data but suffered from noticeable overfitting at large C values. Despite the overfitting, the RBF kernel achieved the highest test accuracy among all kernels tested, suggesting that nonlinear decision boundaries are beneficial for capturing the intricate relationships in house price range prediction when properly regularized.



3.2.7 Conclusion and Table

Based on the experiments conducted with various SVM kernels and transformations, the Support Vector Machine with an RBF (Radial Basis Function) kernel demonstrated the best overall performance for predicting house price ranges. The RBF kernel was able to capture complex nonlinear relationships in the data that linear models and polynomial kernels could not fully model. Although the RBF model showed some degree of overfitting at high regularization parameters, it consistently achieved the highest testing accuracy among all models, peaking at around 35.4% with a C value of 500. This suggests that the flexibility of the RBF kernel in mapping data into a higher-dimensional feature space was crucial for addressing the overlapping and nonlinear structure of the house price categories.

Transformation	C	Train Accuracy	Test Accuracy	Precision	Recall
No Transform	0.01	0.25679315	0.24473813	0.19147568	0.24473813
No Transform	0.1	0.31064872	0.28438571	0.21199272	0.28438571
No Transform	1	0.33390453	0.31081743	0.26243	0.31081743
No Transform	5	0.33121175	0.30885952	0.26244321	0.30885952

No Transform	10	0.3375765	0.30983847	0.28065783	0.30983847
No Transform	100	0.34296206	0.30885952	0.27796924	0.30885952
Poly Kernel Degree 2	0.01	0.21395349	0.20704846	0.18384434	0.20704846
Poly Kernel Degree 2	0.1	0.27123623	0.25795399	0.18274948	0.25795399
Poly Kernel Degree 2	1	0.35446756	0.30739109	0.26825974	0.30739109
Poly Kernel Degree 2	5	0.39388005	0.32599119	0.30868175	0.32599119
Poly Kernel Degree 2	10	0.40097919	0.33284386	0.31456893	0.33284386
Poly Kernel Degree 2	100	0.44479804	0.32941752	0.30685454	0.32941752
Poly Kernel Degree 3	0.01	0.22643819	0.20998532	0.18560908	0.20998532
Poly Kernel Degree 3	0.1	0.32044064	0.28634361	0.2399769	0.28634361
Poly Kernel Degree 3	1	0.39853121	0.32550171	0.30286784	0.32550171
Poly Kernel Degree 3	5	0.45483476	0.34361234	0.32080401	0.34361234
Poly Kernel Degree 3	10	0.47441861	0.35095448	0.3274879	0.35095448
Poly Kernel Degree 3	100	0.57429621	0.34703867	0.33448644	0.34703867
Poly Kernel Degree 4	0.01	0.2506732	0.21781694	0.18544798	0.21781694
Poly Kernel Degree 4	0.1	0.35911873	0.29025942	0.25603774	0.29025942
Poly Kernel Degree 4	1	0.44773562	0.33382281	0.30764623	0.33382281
Poly Kernel Degree 4	5	0.5255814	0.3485071	0.3297099	0.3485071
Poly Kernel Degree 4	10	0.56695226	0.34459129	0.32887747	0.34459129
Poly Kernel Degree 4	100	0.69326805	0.35291238	0.3445682	0.35291238
Log Transform	0.01	0.15716034	0.15418502	0.04754679	0.15418502
Log Transform	0.1	0.24161567	0.22320118	0.14709197	0.22320118
Log Transform	1	0.3001224	0.28291728	0.21202551	0.28291728
Log Transform	10	0.33684211	0.30885952	0.25600648	0.30885952
Log Transform	100	0.34418605	0.30641214	0.27211029	0.30641214

Log Transform	500	0.34810282	0.31277533	0.27886092	0.31277533
RBF Kernel	0.01	0.14859241	0.15026921	0.05025078	0.15026921
RBF Kernel	0.1	0.23916769	0.22662751	0.13806021	0.22662751
RBF Kernel	1	0.35740514	0.30445423	0.28037469	0.30445423
RBF Kernel	10	0.46168911	0.33578072	0.31117246	0.33578072
RBF Kernel	100	0.59510404	0.34703867	0.33455499	0.34703867
RBF Kernel	500	0.71358629	0.35438081	0.34714326	0.35438081

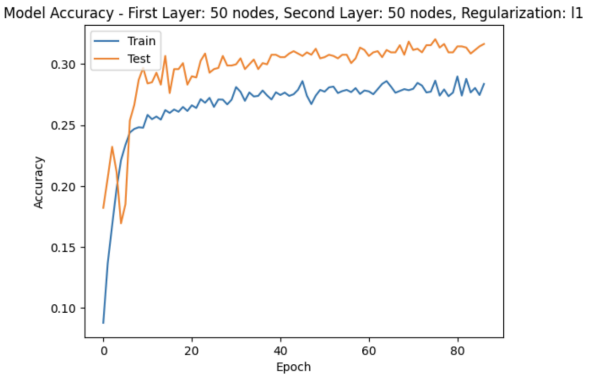
3.3 Neural Networks

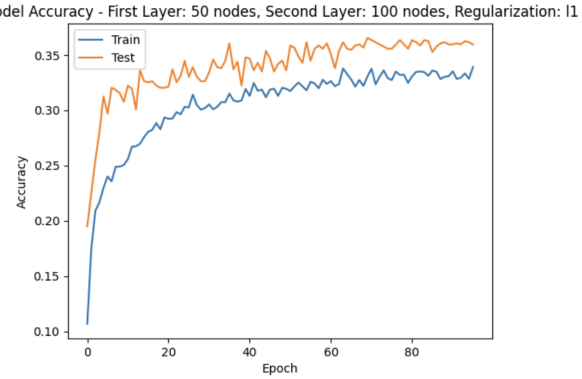
For the third model we used to predict, we choose neural networks because of its high learning capability and deep learning property. The hyperparameters we have here are no transformation, polynomial transformation with degree 2, different nodes in each of 2 hidden layers, and different regularization values. Overall, the best test accuracy with no transformation is 41.63%, with hyperparameters 700 and 600 nodes in first and second hidden layer respectively, L2 regularization with value 0.005. With polynomial transformation of degree 2, the best accuracy is 40.65%, with 500 and 100 nodes in first and second hidden layer respectively, L2 regularization with value 0.01.

We trained 180 hyperparameters' combinations, with each of them running 100 epochs. Initially, we design 144 combinations with first layer node number being selected from [50,200,500], second layer node number from [50,100], regularization type from ['L1', 'L2'] with values from [0.001, 0.005, 0.008, 0.01, 0.05, 0.1], and transformation from ['no transformation', 'transformation with polynomial of degree 2']. Some sample data are listed below:

Test Accuracy	First Layer N	Second Laye	Regularizatic	Regularizatic	Train Accuracy	Train Precision	Train Recall	Train MSE	Train R^2	Val Accuracy	Val Precisior	Val Recall	Val MSE	Val R^2	transformation
0.3741	50	50	l1	0.001	0.336436003	0.443427742	0.375382731	4.387630129	0.7021972	0.3712047	0.4369154	0.3712047	5.4867777	0.6276689	No transformation
0.3751	50	50	l2	0.001	0.333741575	0.446750276	0.3821188	4.311083895	0.7073927	0.3790402	0.4403483	0.3790402	5.122429	0.6523935	No transformation
0.3624	50	50	l1	0.005	0.335823625	0.438101618	0.372320882	4.825842009	0.6724544	0.3643487	0.4295233	0.3643487	5.9010774	0.5995546	No transformation
0.3761	50	50	l2	0.005	0.3402327	0.44668553	0.381383956	4.606368647	0.6873508	0.3731636	0.4359892	0.3731636	5.6062684	0.6195603	No transformation
0.3369	50	50	l1	0.008	0.314023256	0.421022517	0.352602572	5.055725658	0.6568514	0.3369246	0.4012691	0.3369246	5.7002938	0.6131797	No transformation
0.3751	50	50	l2	0.008	0.33349663	0.4512417	0.38922229	4.308266993	0.7075839	0.3780607	0.44305	0.3780607	5.4809011	0.6280677	No transformation
0.35847208	50	50	l1	0.001	0.33570117	0.44637629	0.38077159	4.37391304	0.7031283	0.3584721	0.4281266	0.3584721	5.3095005	0.6396989	Poly degree 2 transformation
0.36728698	50	50	l2	0.001	0.33239436	0.446676684	0.3816289	4.50226577	0.6944166	0.3633693	0.4258445	0.367287	5.3927522	0.6340494	Poly degree 2 transformation
0.36336926	50	50	l1	0.005	0.33619106	0.43481646	0.37134109	4.75701164	0.6771261	0.3682664	0.4273608	0.3633693	5.6797258	0.6145755	Poly degree 2 transformation
0.3653281	50	50	l2	0.005	0.33570117	0.40256299	0.38922229	4.30802205	0.7076005	0.3712047	0.428506	0.3653281	5.4711068	0.6287323	Poly degree 2 transformation
0.33888346	50	50	l1	0.008	0.3195346	0.42207851	0.35260257	4.86858543	0.6695532	0.3369246	0.3966171	0.3388835	5.662096	0.6157718	Poly degree 2 transformation
0.3623898	50	50	l2	0.008	0.32884261	0.40715174	0.37917942	4.48560931	0.6955471	0.3653281	0.4257493	0.3623898	5.4417238	0.6307262	Poly degree 2 transformation
0.34573948	50	50	l1	0.01	0.31708512	0.42288618	0.35823637	4.86981017	0.6694701	0.337904	0.4073429	0.3457395	5.5523996	0.6232158	Poly degree 2 transformation
0.3770813	50	50	l2	0.01	0.33251685	0.44412973	0.3826087	4.59791794	0.6879243	0.3721841	0.4356306	0.3770813	5.5778648	0.6214877	Poly degree 2 transformation

Then we find the model is experiencing strong underfit when there are small amounts of nodes. When using 50 or 100 nodes each layer, test accuracy is always greater than train accuracy, for example:



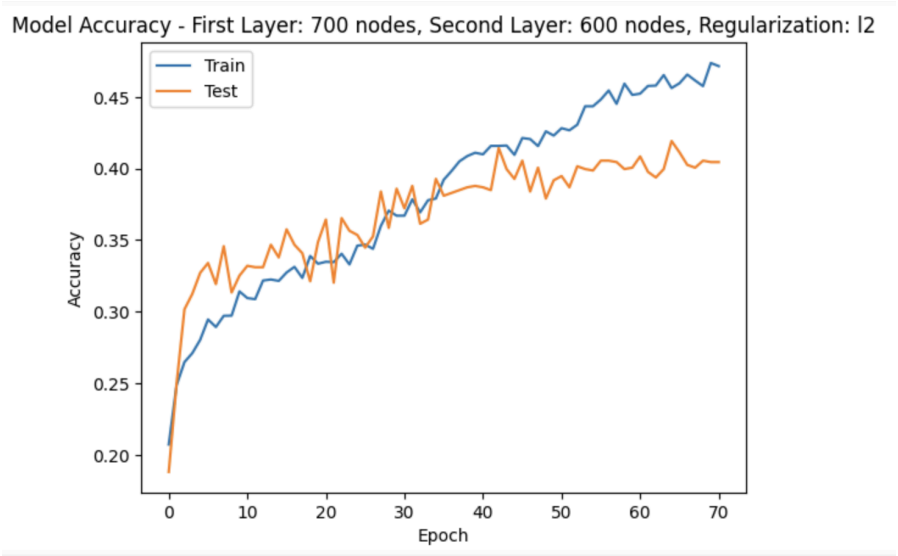


Thus, we design another 36 combinations with first layer node number from [700, 900, 1000] and second layer node from [600,800], using just L2 regularization because L1 regularization does not help increase accuracy before. Some examples of improved transformation are:

Test Accuracy	First Layer N	Second Laye	Regularizati	Regularizati	Train Accuracy	Train Precision	Train Recall	Train MSE	Train R^2	Val Accuracy	Val Precisior	Val Recall	Val MSE	Val R^2	transformation
0.383937	700	600	l2	0.1	0.376118	0.404953	0.4109	4.203674	0.714683	0.38002	0.406378	0.383937	5.468168	0.628932	No transformation
0.399608	700	800	l2	0.001	0.436252	0.495798	0.493815	3.422535	0.767701	0.390793	0.383217	0.399608	4.730656	0.678979	No transformation
0.404505	700	800	l2	0.005	0.482547	0.534119	0.531782	2.836497	0.807478	0.402547	0.410563	0.404505	4.246817	0.711812	No transformation
0.400588	700	800	l2	0.008	0.469565	0.527271	0.52872	2.991917	0.796929	0.397649	0.396908	0.400588	4.874633	0.669209	No transformation
0.416259	700	800	l2	0.01	0.469933	0.51653	0.520514	3.065279	0.79195	0.410382	0.419212	0.416259	4.392752	0.701909	No transformation
0.399608	700	800	l2	0.05	0.401837	0.433993	0.440661	3.802327	0.741924	0.394711	0.392524	0.399608	5.176298	0.648738	No transformation
0.377081	700	800	l2	0.1	0.369994	0.446217	0.401225	4.261849	0.710734	0.375122	0.418902	0.377081	5.608227	0.619427	No transformation
0.398629	900	600	l2	0.001	0.45793	0.514266	0.516105	3.193754	0.78323	0.401567	0.40083	0.398629	4.874633	0.669209	No transformation
0.406464	900	600	l2	0.005	0.475811	0.532444	0.527373	2.961053	0.799024	0.410382	0.402181	0.406464	4.824682	0.672599	No transformation
0.408423	900	600	l2	0.008	0.47177	0.533156	0.524434	2.997795	0.79653	0.405485	0.414297	0.408423	4.53477	0.692272	No transformation
0.398629	900	600	l2	0.01	0.46834	0.522917	0.518677	3.028904	0.794418	0.400588	0.393377	0.398629	4.628795	0.685891	No transformation
0.404505	900	600	l2	0.05	0.407961	0.443252	0.437844	3.836497	0.739604	0.398629	0.395797	0.404505	5.126347	0.652128	No transformation

The model becomes less underfitting and doesn't have much overfitting, and has higher accuracy and precision.

For example, this is the best performance we get:

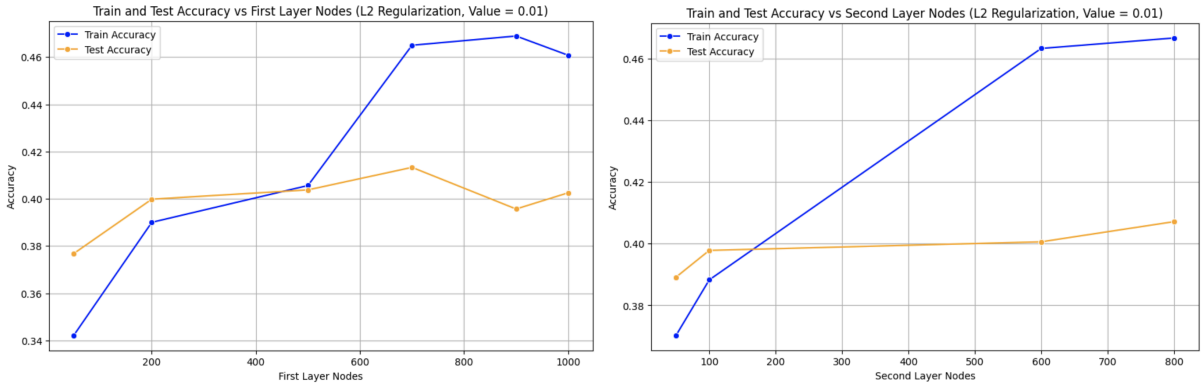


The train accuracy is lower than test accuracy at first, representing underfitting, and then it goes greater than test accuracy with test accuracy becoming steady - the model starts showing a sign of overfitting.

The data is too long to put here, so we store the result of each combination, including Train Accuracy, Train Precision, Train Recall, Train MSE, Train R square, Val Accuracy, Val Precision, Val Recall, Val MSE and Val R square. in the file "Neural Network total output.xlsx".

3.3.1 Effect of number of nodes in each layer

The effect of amount of nodes in the first layer and second layer can be visualized as the following:

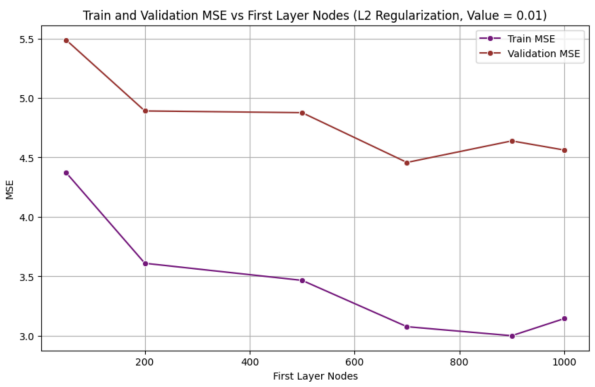


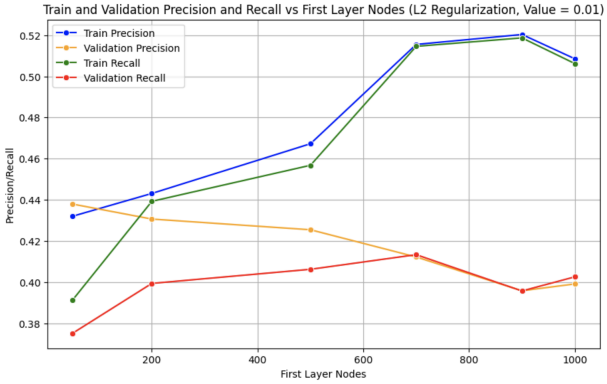
The first shows that with about 200 nodes in the first layer, the model performs a sign of underfitting because the test accuracy is slightly higher than train accuracy, and both accuracies are low. The increasing number of nodes in both layers helps improve the accuracy of the model, because both test and train accuracy are increasing.

When there are about 600 nodes, the model begins to have a sign of overfitting. At about 900 nodes, the sign of overfitting becomes more noticeable because the test accuracy starts to decrease, while training accuracy stays to be high. Similar performance happens in the graph of second layer nodes, while the sign of overfitting starts earlier at about 200 nodes. The overfit can be caused by having too many nodes in each layer so the model trains the training data too well, and the model is too complicated to fit test data. The underfit can be caused by having a model not complicated enough to train data well.

It can be seen from this graph that the model is performing underfitting when there are less nodes in the first and second layer (like 500-50 nodes) because the accuracy of the training set is smaller than that of the test set. It means the model is still learning to get better performance. But

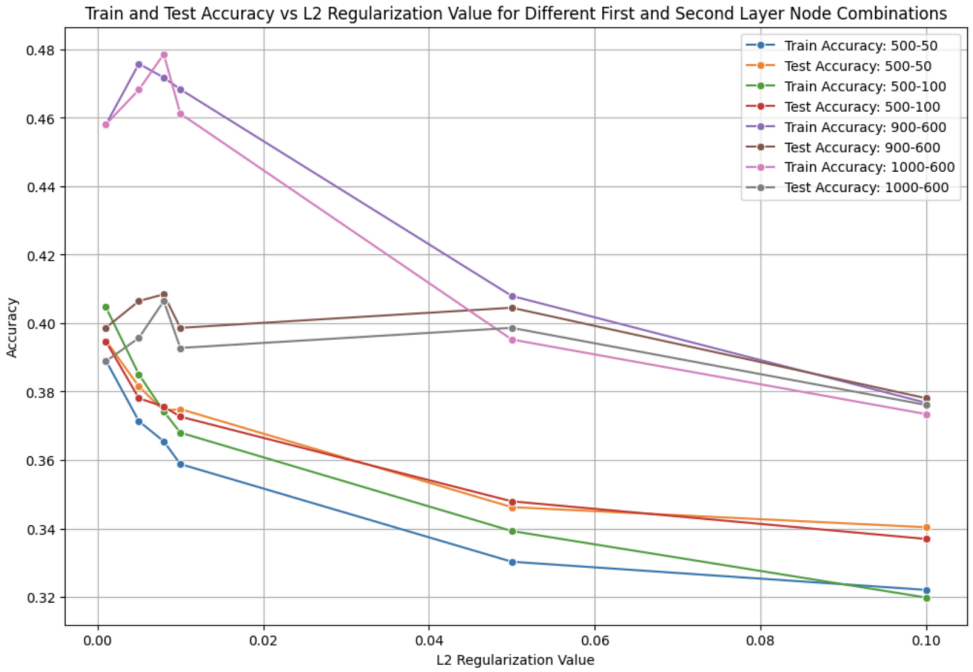
when we have more nodes, accuracy is improved, which matches the finding we have above when changing nodes number in both layers.





The decrease of MSE with increase of layer nodes until about 700 nodes shows that the model is improving its performance, while the slight increase on val MSE and almost no change on training MSE after 700 nodes can be a sign of overfitting. Similar signs of overfitting can also be seen in the figure of precision and recall, since they are getting smaller after 700 nodes.

3.3.2 Effect of L2 regularization



The data output file shows that overall, L1 regularization doesn't perform well in this model since the average accuracy of models with L1 regularization is lower than that with L2 regularization. So we choose to analyze models with L2 regularization here.

Since there is too much data, we only plot some combinations of first and second layer node numbers with no transformation here as an example to show this effect. We find that L2 regularization value's increment doesn't perform well during this model training, because both the train accuracy and test accuracy increases when L2 regularization value is small, but continues decreasing when the value becomes larger. Thus we find that higher regularization values are penalizing the model too much, which leads to underfitting.

3.3.3 Effect of polynomial of degree 2 transformation

Since now we have enough number of nodes in each layer to avoid underfitting, we choose to apply the polynomial of degree 2 transformation to the data because with large amount of nodes in each layer, the model is becoming more powerful in handling high dimensional data and capture nonlinear relationships in data.

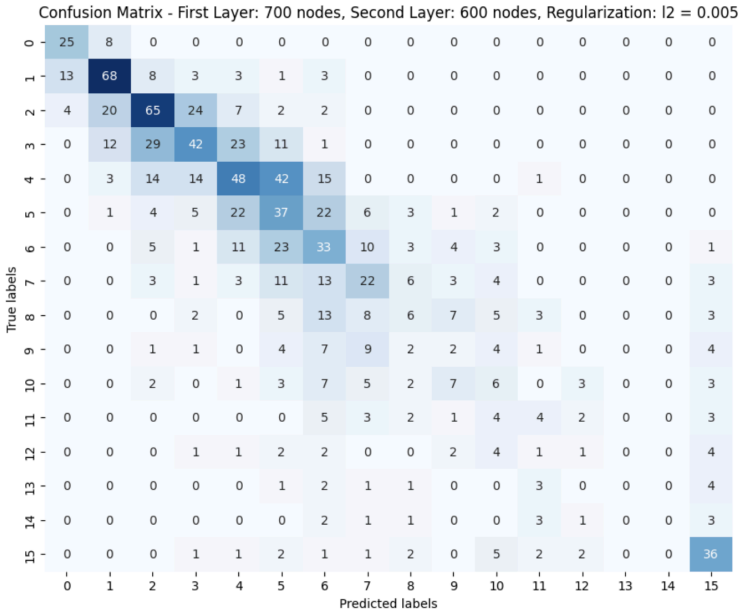
We calculate the average accuracy, validation precision, validation MSE and validation recall.

	Average accuracy	Average Val Precision	Average Val Recall	Average Val MSE
No transformation	0.363030556	0.416042239	0.370238946	5.472695771
Poly degree 2 transformation	0.362865926	0.413857301	0.367885515	5.4983132

Applying a polynomial of degree 2 transformation has slightly lower accuracy, precision and recall and a slightly higher MSE. This means applying polynomial transformation makes the model performance worse, and keeping no transformation to the data can be better.

3.3.4 Conclusion of neural networks model

We generates the confusion matrix of neural networks model below:



By observing the diagonal, we see that labels with values 0-7 and 15 are predicted well, but for labels 8-14, more wrong predictions appear. Overall, the neural networks model performs better since this figure shows we have more true predictions, and for those wrong predictions, predicted labels are getting closer and more concentrated to the true label.

In conclusion, the performance of the neural networks model is better than the previous 2 models because with hyperparameters of 700 and 600 nodes in the first and second hidden layer respectively and a L2 regularization with value 0.005, the model can get an accuracy of 41.63%. It also has val precision 40.63% and val recall 41.63% which are higher than previous models, and MSE 4.8 which is lower than previous models. This hyperparameters' combination gets the best performance among all the models we trained.

4. Conclusion

In order to make predictions about the price range of a house in California, we do data cleaning first to ensure that the dataset does not have null value and is not highly unbalanced. For unsupervised analysis, we use PCA and clustering to help us reduce data collinearity and bias.

4.1. Models summary

The table below shows the best test accuracy, precision and recall we get for each model.

	Linear regression	SVM	Neural Networks
Best test accuracy	28.34%	35.40%	41.63%
Best precision	28.80%	34.70%	40.63%
Best recall	28.34%	35.40%	41.63%
Hyperparameters used	Polynomial degree 3 L2 regularization Alpha = 0.01	RBF Kernel L2 regularization C = 500	700 nodes in hidden layer 1 600 nodes in hidden layer 2 No transformation L2 regularization with value 0.005

Why we choose each model and how we train them:

We initially used linear regression for the house price prediction task because the target variable is continuous. After training the model with a polynomial degree 2 and 3 and doing L2 regularization, the accuracy of the model gets slightly improved. But this approach still produces poor results, with low accuracy (around 20%) and high MSE, indicating underfitting. This led us to conclude that linear models were insufficient for capturing the complexity of this task. In order to get a better prediction, we decided to explore nonlinear models like Support Vector Machines (SVM). SVM can be better suited for handling complex, nonlinear relationships in the data.

The SVM model successfully helps reduce underfit, especially when we train it with an RBF kernel due to the large number of features we have. With the scale gamma value and high C regularization value, the training accuracy can reach to about 71%, meaning that it fits the training examples well compared to the linear regression model. However, the testing accuracy around 35% shows that the model is overfitting with the high flexibility. To better handle this complicated dataset, we choose neural networks to be our third model.

For Neural networks model, we train the model with different number of nodes in first layer (50, 200, 500), in second layer (50, 100), different regularization types (L1, L2), different regularization value (0.001, 0.005, 0.008, 0.01, 0.05, 0.1), and apply polynomial transformation with degree 2 or not. We train 144 models with these combinations, and have our best accuracy as 41.33%, with 40.74% validation recall and 42.22% validation precision when we have 200 first layer nodes, 100 second layer nodes and L2 regularization with value 0.01. Then when we

analyze the train accuracy & test accuracy v.s. epochs diagram, we find that with a small number of nodes, the model is strongly underfitting since the train accuracy is continuously lower than test accuracy. Thus, we create another 36 combinations of hyperparameters to train the model which includes first layer node number from [700, 900, 1000] and second layer node number from [600, 800]. The performance of the model now becomes better, with the highest test accuracy 41.63%.

In summary, we analyze and observe the sign of overfit and underfit, step by step determine the goal of the next model, add and improve our transformations and analyze the data, and finally train a model with a neural network that has flexibility to deal with data complexity.

4.2 Reflection and Future Work

We think the reason for the low accuracy of our model may be that the imbalance in the distribution of the data has a bigger impact than we thought. Since the data distribution (number of houses v.s. price range histogram) appears to be left-skewed, houses with lower price range are more than those with larger price range. The model might become biased toward predicting the more frequent classes, so it predicts less accurately for those with high true label houses (like true label 8-14).

A possible solution is to use SMOTE to balance the dataset by forming synthetic samples for those classes with smaller amounts of samples. While generating these new samples, we need to pay attention to possible overfits to let the model not focus too much on those synthetic data.

The other result that can be improved is the training of neural networks models. Since the model has very high flexibility, people can use a more powerful computer to test more combinations of hyperparameters to find an optimized result of layer numbers, number of nodes in each layer, transformations, regularization type and value, etc. The things we did is to train, observe, and train again using better transformations and hyperparameters to get a better result, but with a more powerful computer, this model can be improved a lot.