To Flip or Not To Flip?

Abstract

Home flipping is a rising industry with a low barrier cost of entry that allows both investors and homeowners to grow personal wealth. The process of searching for the best candidate house to flip is extensive, but can be simplified by viewing information about house listings online using real estate marketplace websites, such as Zillow or Redfin. This paper attempts to develop an effective machine learning model to classify house listings as flippable or not flippable. Using scraped house listing data from Zillow, we build three classification models: KNN, logistic regression, and SVM. We then optimize each model using hyperparameter tuning. We find that the logistic regression model had the best performance before and after optimization, while the KNN model improved the most during optimization. However, none of the three models showed any high level of accuracy.

Introduction

Home flipping is a popular real estate investment strategy in which investors purchase properties to later sell for a profit. This strategy often involves finding properties in poor condition, completing any necessary repairs and renovations, and then selling for a profit within a short amount of time. One of the biggest challenges for home flippers is choosing the right property to flip. Properties must show high potential for profit based on multiple metrics, such as condition, physical characteristics, and neighborhood. Traditional home flippers often do not have access to important property data, and instead must visit multiple properties to assess each and find the most profitable candidate for flipping. This process is time-consuming and expensive. By utilizing public data and machine learning models, the process of choosing a property to flip can be made significantly more efficient.

In this project, we aim to build a machine learning model that accurately predicts if a property should be flipped or not. We use data scraped from Zillow, a prominent online real estate marketplace, and available demographics data to build this model.

Technical Approach

The objective of this project is to build an effective model that predicts if a home will be a profitable flip based on various features of the property. The model will take in feature input data and then classify the property as flippable or not flippable. We have chosen three classification models to construct and compare the performance of: k-nearest neighbors, logistic regression, and support vector machines.

K-Nearest Neighbors

The k-nearest neighbors (KNN) classification algorithm is based on the idea that similar data points are close in distance to one another. The KNN model predicts the class of a data point based on the classes of the points nearest to it. Of the K number of neighbors nearest to the data point, the class that the majority of neighbors belong to becomes the predicted class of the data point. The algorithm can be summarized as follows:

Step 1. Initialize K, the chosen number of neighbors.
Step 2. Calculate the distance between the data point and all training data samples. *Step 3.* Select the K number of data samples closest in distance to the data point.

Step 4. Assign a class to the data point based on the class that the majority of the K nearest neighbors have.

Step 5. Repeat steps 2-4 for each data point.

There are several different methods for calculating the distance between data points. The most popular measure is Euclidean distance, which represents the shortest distance between two points. For an n-dimensional space, it is defined as: $f.(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

Another commonly-used measure is Manhattan distance, which is the sum of the absolute distance $(x,y) = \sum_{i=1}^{n} x_i -$ between two points across all dimensions. For an n-dimensional space, it is the hyperparameter tuning section of the paper.$f \quad | \quad y |$.

defined as:We will explore the performance of both distance measures in $_i$

Logistic Regression

Binary logistic regression is a statistical model used to predict the likelihood of a categorical binary dependent variable. The model utilizes the sigmoid function to map predicted values to a probability between 0 and 1. The sigmoid function is defined as: $S(z) = {}_{1+1\,e_{-z}}$, where z is the predicted value by the model. Using the probability returned by the sigmoid function, the model classifies the input into one of the two output classes based on the decision boundary. When the threshold value of the decision boundary is 0.5, a data point will be predicted to be class 0 if the probability is less than 0.5 and class 1 if the probability is equal to or greater than 0.5.

Logistic regression models often use a regularization term to prevent overfitting on the data. Two common terms are L1 regularization and L2 regularization. The L1 regularization term is defined as the sum of the absolute value of the coefficients. L1 limits the size of the coefficients and allows some coefficients to become zero and be eliminated. This means that L1 may yield sparse models. The L2 regularization term is defined as ½ of the sum of the square of the coefficients. L2 shrinks all the coefficients by the same value and will not eliminate any coefficients. We will explore the performance of each regularization term in the hyperparameter tuning section.

Support Vector Machine

The support vector machine (SVM) algorithm finds a hyperplane in an n-dimensional space that separates the data points of each class. Since there are many possible hyperplanes that can separate the data, the objective is to find the hyperplane with the maximum margin, or the maximum distance between the data points of each class. To find the maximum margin, the algorithm uses support vectors that determine the position of the hyperplane. Support vectors are the data points closest to the hyperplane and determine

the position of the hyperplane. Adding or removing support vectors changes the hyperplane, so the support vectors are the points that build the SVM.

SVMs often use kernel functions to transform non-linearly separable data into a higher-dimensional space, making the data linearly separable. There are several different types of kernel functions. One of the most used functions is the Gaussian radial base function (RBF) kernel function, which is defined as: $K(X_1, X_2) = exp(-|X_1 - X_2|_2)$. RBF kernels are often used when there is no prior knowledge about the data. Another popular kernel is the linear kernel, which is defined as: $K(X_1, X_2) = X_{T1}X_2$. Linear kernels are the most basic form of a kernel and are often used when the data is already linearly separable. We will explore the performance of both kernel functions in the hyperparameter tuning section.

2σ

Experimental Results

Data Collection and Target Creation

To collect real estate data, we created a scraper for Zillow. The scraper scrapes all available listings from a specific view of the map, as well as each listing's basic features, such as area, price, bedrooms, and bathrooms. By using a specific view of the map, listings are restricted to a small area of locations, so the data does not include suburbs that may lack access to schools and other amenities that areas closer to the city may have.

After collecting the Zillow data, we integrated more data into each listing by filtering by zip code. We used a demographics dataset that contained valuable information by zip code, such as unemployment rate, poverty rate, education level, and median household income. We integrated these features into the dataset to gain more insight into the house and the surrounding area.

To create the target variable, we manually classified each listing as flippable or not. To do this, we looked through the photos of the homes in each individual listing and considered all features of the data sample. For example, a house that is close to the highway, not priced too high for the area, and is in need of renovation work would be considered a desirable flip.

Data

Variable	Description
price	Listed price of the house ($)
bathrooms	Number of bathrooms
bedrooms	Number of bedrooms
area	Area of the house (sqft)
zestimate	Zillow's estimated value of the house ($)
rent_zestimate	Zillow's estimated rent/month of the house ($)
days_on_zillow	Number of days listed on Zillow
$/sqft	Price per sqft of area of the house ($/sqft)
compare_$sqft_sold_inZip	Percentile of $/sqft compared to other houses sold in the same zip code
	Percentage difference between $70,000 and the

household_median_income_on_threshold	median household income of the zip code
median_age	Median age of the zip code
median_house_value	Percentage difference between the house's value and the median house value in the same zip code
poverty_rate	Poverty rate of the zipcode
unemployment_rate	Unemployment rate of the zipcode
target	1 if flippable, 0 if not flippable

Table 1. Variable descriptions.

Our original dataset had 14 feature variables and one target variable. Table 1 summarizes each variable. After cleaning to remove any rows with null values, the dataset had 489 samples. There were 214 samples in the flippable class (1) and 275 samples in the not flippable class (0).

Correcting for Multicollinearity

While initially exploring the data, we created a pairplot to visualize the relationships between each of the feature variables. We noticed that several of the variables showed a high level of correlation with one another. This prompted us to test and correct the data for multicollinearity.

To measure the level of correlation among the feature variables, we used the measure of variance inflation factor (VIF), which denotes higher values for higher levels of correlation. To calculate VIF, we used the *variance_inflation_factor()* function that is available in the statsmodels library.

Feature	VIF (14)	VIF (12)
price	13.046	2.272
bathrooms	1.619	1.600
bedrooms	1.393	1.391
area	7.129	2.121
zestimate	8.743	
rent_zestimate	1.299	1.264
days_on_zillow	1.060	1.045
$/sqft	6.512	
compare_$sqft_sold_inZip	1.435	1.349
household_median_income_on_threshold	3.185	3.161
median_age	1.061	1.056
median_house_value	3.083	2.969
poverty_rate	2.121	2.118
unemployment_rate	1.270	1.263

Table 2. VIF values for all 14 feature variables and for the 12 selected feature variables.

The second column of Table 2 shows the VIF for each of the 14 feature variables. The variables price, area, zestimate, and $/sqft have high VIF values, meaning they are highly correlated with other feature variables. To correct for this, we decided to remove the variables zestimate and $/sqft in order to reduce the correlation of the other variables. The last column of Table 2 shows the VIF values for the feature variables after removing zestimate and $/sqft. The variables show reasonable low VIF values, meaning that there is not a problem of multicollinearity with these 12 selected feature values.

Model Implementation and Training

We implemented the KNN, logistic regression, and SVM classification models using their respective classes from the scikit-learn library. Since we performed hyperparameter tuning later on, we initially implemented the models using the default parameters set by scikit-learn. To train the models, we fit each model using the training set of the 12 selected feature values and the training set of the target values. The training set and testing set were split using the *train_test_split()* function from scikit-learn.

Results of the initial implementation of the three classification models show that the logistic regression model had the best overall performance, with 57.1% accuracy for training and 62.6% accuracy for testing. The SVM model had the second best overall performance with 56.6% accuracy for training and 55.3% accuracy for testing. Although the KNN model had the best training performance with 71.6% accuracy, the model seems to have overfitting problems due to an accuracy score of only 44.7% for testing.

Hyperparameter Tuning

For each model, we performed hyperparameter tuning using scikit-learn's GridSearchCV class. GridSearchCV takes in a parameter grid and determines the highest performing combination of parameters using cross validation. We used cross validation with 3 folds for both accuracy and optimal performance.

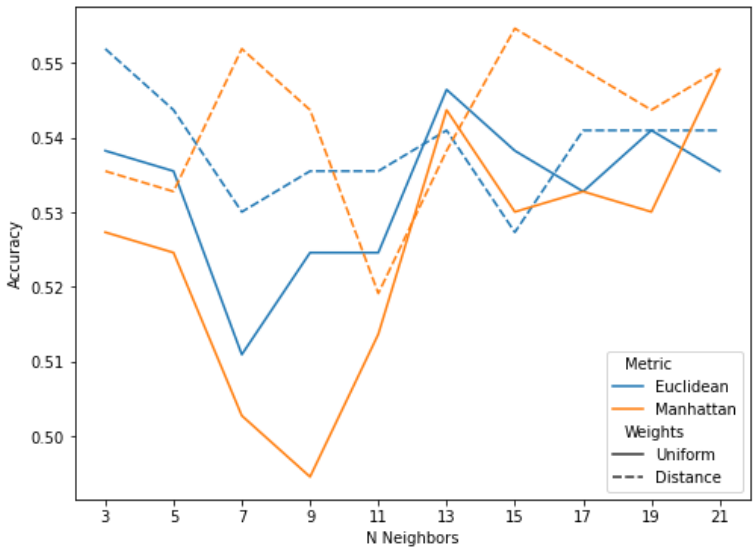

Figure 1. Hyperparameter Tuning Accuracy Scores for the KNN model.

We chose to tune three parameters for the KNN model: n_neighbors, metric, and weights. The n_neighbors parameter is important to tune because it determines the number of neighbors that are considered when classifying a data point. We considered every odd

number in the range of 3 to 21 in order to provide a large range of neighbors. The metric parameter determines the metric used to calculate the distance between two points. We considered Euclidean distance and Manhattan distance. Lastly, the weights parameter determines if the model gives different weight to data points based inversely on distance. We considered a uniform (non-weighted) model and a distance-weighted model.

The grid search found that the best combination of parameters for this model is 15 neighbors with Manhattan distance and a distance-weighted model. This combination had a cross-validation accuracy score of 55.4% and a test score of 50.4%. By performing hyperparameter tuning on the model, we were able to significantly reduce the problem of overfitting that we had on our initial model. Figure 1 shows the accuracy for each combination of the tuning parameters. When less neighbors are used, a distance-weighted model tends to outperform non-weighted models. However, as more neighbors are used, weight does not seem to have a significant effect on performance. In addition, the Manhattan distance metric seems to vary more based on if the model is distance-weighted or not, while the Euclidean distance metric does not seem to be affected as much.

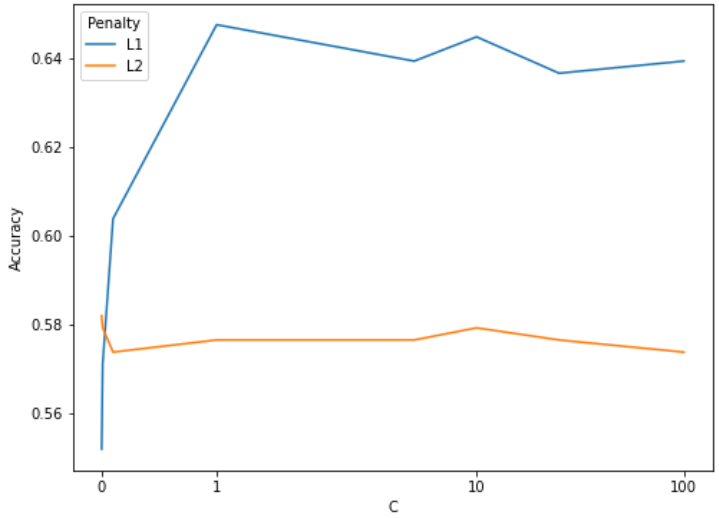

Figure 2. Hyperparameter Tuning Accuracy Scores for the Logistic Regression model.

We chose to tune two parameters for the logistic regression model: penalty and C. The penalty parameter determines the regularization term used. We considered both the L1 regularization term and the L2 regularization term. The C parameter determines the regularization strength, where smaller values specify stronger regularization. We considered the values 0.001, 0.01, 0.1, 1, 5, 10, 25, and 100 for a wide range of C values.

The grid search found that the combination of L1 regularization and a C value of 1 had the best performance with a cross-validation accuracy score of 64.8% and a test accuracy score of 54.5%. Figure 2 shows the accuracy scores for each of the combinations of the parameters. Overall, the L1 regularization term had significantly higher accuracy than the L2 regularization term. However, when regularization is very strong at small C values, L1 does not perform as well and has lower accuracy than L2.

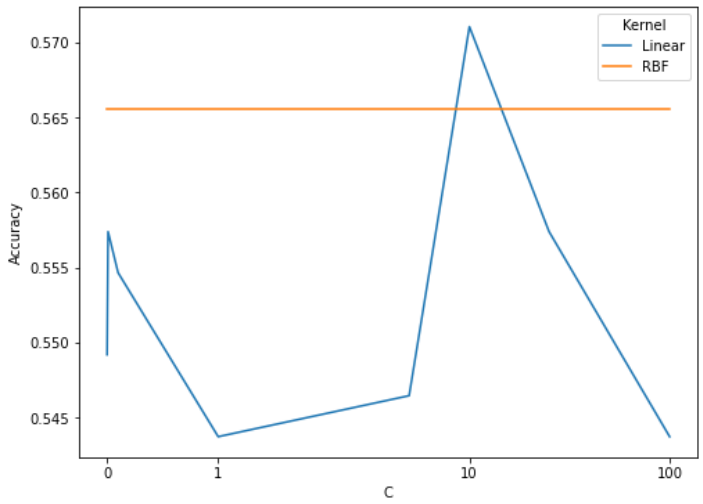

Figure 3. Hyperparameter Tuning Accuracy Scores for the SVM model.

For the SVM model, we chose to tune two parameters: kernel and C. The kernel parameter determines the kernel function that is used in the model algorithm. We considered the linear kernel function and the RBF kernel function. The C parameter is similar to the C parameter in the logistic regression model, in that it inversely determines the regularization strength. We again considered the values 0.001, 0.01, 0.1, 1, 5, 10, 25, and 100 for C.

The grid search found that the linear kernel function with a C value of 10 resulted in the best performance. This combination had a cross validation accuracy score of 57.1% and a test accuracy score of 61.0%. Figure 3 shows the accuracy scores for each combination of tuning parameters. Overall, the linear kernel function tended to have lower accuracy than the RBF function. The linear kernel also did not seem to have an established pattern for changes in C as the accuracy scores varied randomly. On the other hand, the RBF kernel was not affected by changes in C and had the same accuracy score for every value of C.

Results

Model	Accuracy before optimization	Accuracy after optimization
KNN	44.71%	51.22%
Logistic Regression	62.6%	61.78%
SVM	55.29%	58.53%

Table 3. A comparison of performance accuracy before and after model optimization.

Table 3 shows a comparison of performance accuracy before and after model optimization. The results show that the models are overall poor at predicting if a house should be flipped or not. Much of the error can be attributed to the manual selection process of the target variable, as well as a lack of inclusion of pictures in the target variable. For

example, a major portion of the decision making in home flipping is based on pictures, and one person could be very biased about which house is truly flippable.

Performing model optimization on the three classification methods helped increase the accuracy only marginally. The accuracy scores after optimization are the mean of the 3-fold cross validation, while the accuracy scores of the initial models are simply the accuracy score of the testing set. Therefore, we can see that the only substantial improvement from optimization is of the KNN model, which increased accuracy by about 6%. Optimization of the SVM model only slightly improved accuracy by about 3% and optimization of the logistic regression model did not affect accuracy to a large extent. However, the logistic regression model continued to have the highest accuracy score both before and after optimization.

For further work, we can improve our approach by incorporating pictures into the model. Since home flippers generally look at pictures to determine if the house needs renovation work, it would be helpful to feed the model pictures of the home as well. This would help reduce bias in certain edge cases that will help the model better classify houses with good features but higher priced houses, or bad features but good priced houses.