STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Classification of houses based on Price Range using Neural Networks and Tree Classifiers from AHS 2017 Dataset

Course Name:
CPE - 695 Applied Machine Learning

Professor:
Prof. Shucheng Yu

Team Members:

Darp Raithatha
Likith Nandigam
Monami Mukhopadhyay

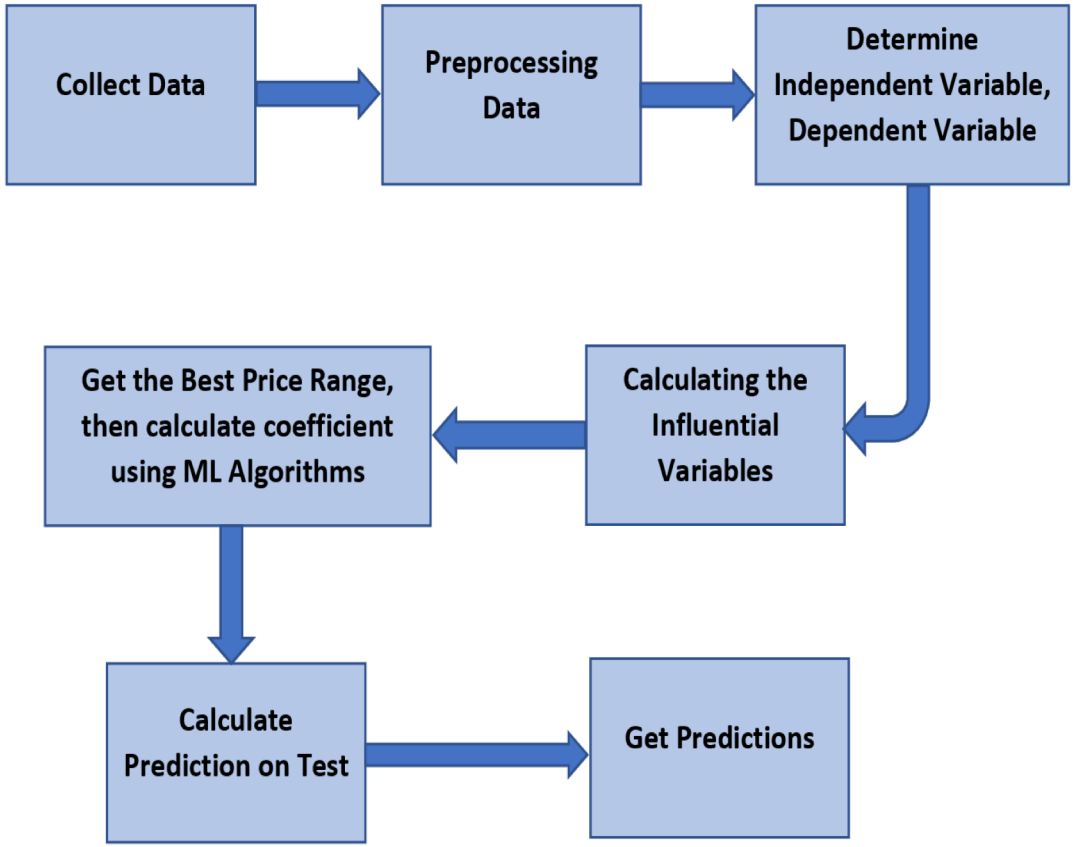
Problem Description

Problem:

- The right price to sell a house
- Having a computer algorithm do this is a more optimized way of doing this

Solution:

- The main goal of this project is to predict the range of selling price of house with a high degree of predictive accuracy using various Machine Learning methods.
- Given house sale data or explanatory variable such as number of bedrooms, number of bathrooms in unit, housing cost, annual commuting cost etc., we build our model.
- The model is then evaluated with respect to test data, and plot the prediction and coefficients. In this way, the value of a house can be calculated in a given geographical area.



Dataset Description

Dataset

- American Housing Survey 2017 data (household.csv in AHS 2017 National PUF v3.0 CSV.zip)
- RangeIndex: 66752 entries, 0 to 66751
- Columns: 1090 entries

Data Exploration

- Some of the features can be summarized as:

  MARKETVAL (target column) – Current market value of unit

  PROTAXAMT – Monthly property tax amount

  BEDROOMS – Number of bedrooms in unit

  HINCP - Household Income (past 12 months)

```
data['MARKETVAL'].describe()
```

```
count    3.995100e+04
mean     3.382110e+05
std      5.246493e+05
min      1.000000e+00
25%      1.210225e+05
50%      2.265120e+05
75%      3.958310e+05
max      9.999998e+06
Name: MARKETVAL, dtype: float64
```

CONTROL	TOTROOMS	PERPOVLVL	COMTYPE	COMCOST	JACPRIMARY	JACSECNDRY	JADEQUACY	JBATHEXCLU	JBATHROOMS
'11000001	8	501	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000002	7	501	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000005	8	501	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000006	5	361	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000007	8	501	'1'	5564	'0'	'0'	'2'	'0'	'0'
'11000008	5	114	'-6'	-6	'0'	'0'	'1'	'0'	'1'
'11000009	6	-6	'-6'	-6	'0'	'0'	'1'	'0'	'1'
'11000010	7	20	'3'	278	'0'	'0'	'2'	'0'	'0'
'11000012	3	231	'-6'	-6	'0'	'0'	'1'	'0'	'0'
'11000013	8	310	'1'	2226	'0'	'0'	'2'	'0'	'0'
'11000015	6	385	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000016	6	501	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000017	7	501	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000018	3	158	'-6'	-6	'0'	'0'	'2'	'0'	'0'
'11000019	3	47	'-6'	-6	'0'	'0'	'1'	'0'	'0'

Data Handling

Data Cleaning

- It is necessary to clean a dataset to make it free from erroneous or irrelevant data; by filling up missing values, removing rows and reducing data size - the final dataset was 36358 rows X 526 columns

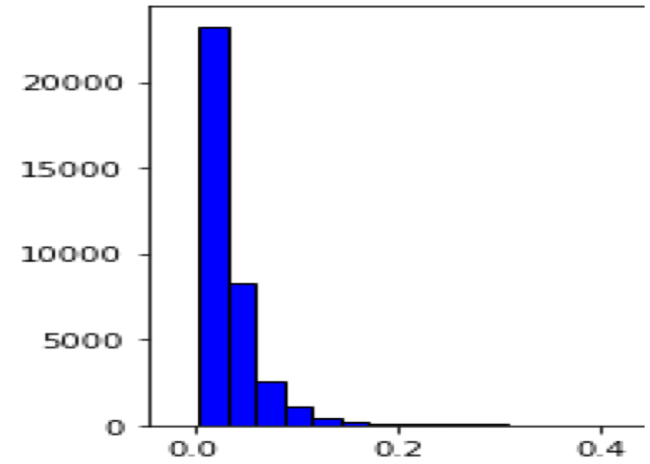
Feature Observation

- All features were evaluated to reduce the dimensionality of the dataset and to check which amongst all features is the most correlated feature for price prediction.

Refine Data

- Critical analysis of the data was necessary to derive relevant statistics. Then the data was then checked for inconsistencies and removed accordingly.

	TOTROOMS	PERPOVLVL	COMCOST	DINING	LAUNDY	STORIES	COMDAYS
TOTROOMS	1	0.2	0.021	0.58	0.24	0.1	-0.035
PERPOVLVL	0.2	1	0.024	0.089	0.087	0.13	-0.0015
COMCOST	0.021	0.024	1	0.012	0.0072	-0.0097	0.086
DINING	0.58	0.089	0.012	1	0.048	0.061	-0.013
LAUNDY	0.24	0.087	0.0072	0.048	1	-0.045	-0.013
STORIES	0.1	0.13	-0.0097	0.061	-0.045	1	-0.016
COMDAYS	-0.035	-0.0015	0.086	-0.013	-0.013	-0.016	1
DIST	0.022	0.027	0.69	0.016	0.016	-0.016	-0.0059
RATINGHS	0.12	0.05	0.0012	0.063	0.068	0.024	-0.029
RATINGNH	0.11	0.049	0.0014	0.057	0.053	0.033	-0.018
CELLPHONE	0.21	0.16	0.011	0.078	0.063	0.025	0.013
WEIGHT	-0.036	-0.054	-0.023	-0.051	0.056	-0.028	-0.00089
SP1WEIGHT	-0.013	-0.02	-0.022	-0.02	0.024	-0.0096	0.00047
SP2WEIGHT	-0.019	-0.026	4.4e-17	-0.028	0.034	-0.016	1.6e-16
HHAGE	-0.063	-0.21	-0.034	0.0044	-0.027	-0.046	-0.084

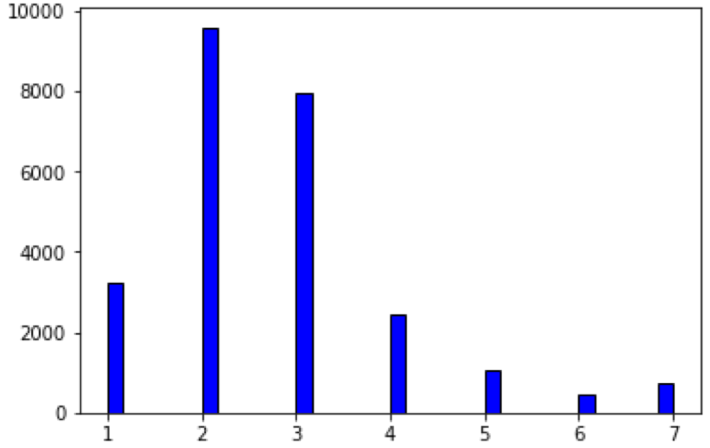


Feature Encoding

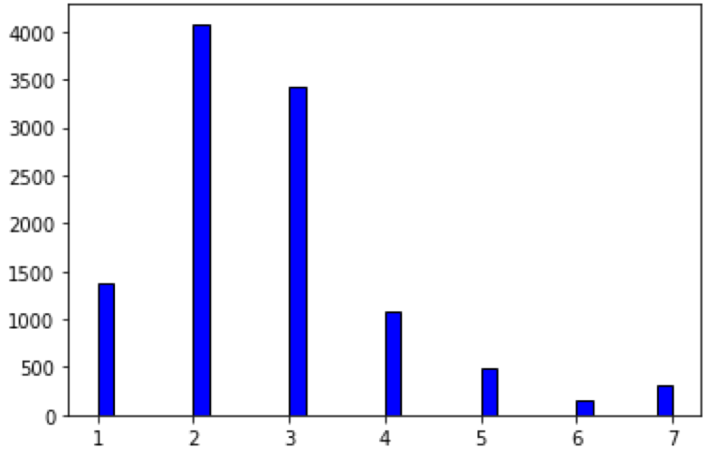
In order to predict the price range for houses, the entire range of MARKETVAL values was divided into smaller ranges and encoded them into set of classes - [1, 2, 3, 4, 5, 6, 7, 13].

```python
# Function to assign code for different price ranges
def price_range(price):
    if price < 100000:
        return 1
    elif price >= 100000 and price < 250000:
        return 2
    elif price >= 250000 and price < 500000:
        return 3
    elif price >= 500000 and price < 750000:
        return 4
    elif price >= 750000 and price < 1000000:
        return 5
    elif price >= 1000000 and price < 1250000:
        return 6
    elif price >= 1250000:
        return 7
    else:
        print(price)
        return 13
```

Train



Test



Random Forest Classifier

```
# Using Random Forest Classifier
model = train_model(X_train, y_train_oh, RandomForestClassifier, n_estimators=200, random_state=20, verbose=True)
accuracy(X_train, X_test, y_train, y_test, model)
# Top 10 features that determine price
pd.Series(model.feature_importances_, x.columns).sort_values(ascending=True).nlargest(10).plot.barh(align='center')
```

The random forest is a classification algorithm consisting of many decision trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.

In order to increase test accuracy, the algorithm was executed using n_estimators = 100, 200, 500, 1000. This resulted in the following accuracies:

- **Training Accuracy – 100.00%**
- **Testing Accuracy – 59.49%**

k-Nearest Neighbors Classifier

```
# Using kNN Classifier
model = train_model(X_train, y_train, KNeighborsClassifier, n_neighbors=7)
accuracy(X_train, X_test, y_train, y_test, model)
```

KNN or k-nearest neighbors is the simplest classification algorithm. This classification algorithm does not depend on the structure of the data. Whenever a new example is encountered, its k nearest neighbors from the training data are examined. Distance between two examples can be the Euclidean distance between their feature vectors. The majority class among the k nearest neighbors is taken to be the class for the encountered example.

For the kNN algorithm, the value for n_neighbors was modified to 3, 4, 5, 6, 7, 8, 9, 10, and finally retained the value that would result in the best accuracy for this use case:

- **Training Accuracy – 60.19%**
- **Testing Accuracy – 47.36%**

Decision Tree Classifier

```
# Using Decision Tree Classifier
model = train_model(X_train, y_train, DecisionTreeClassifier, max_depth=8)
accuracy(X_train, X_test, y_train, y_test, model)
```

Decision Tree Classifier is a simple and widely used classification technique. It poses a series of carefully crafted questions about the attributes of the test record. Each time it receive an answer, a follow-up question is asked until a conclusion about the class label of the record is reached.

The values for max_depth were changed to 2, 3, 4, 5, 6, 7, 8, 9, 10, 20 to finally select the best test accuracy result for Decision Tree:

- **Training Accuracy – 65.08%**
- **Testing Accuracy – 60.60%**

Multi-layer Perceptron Classifier

```
mlp = MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilc
mlp.fit(X_train, y_train)
accuracy(X_train, X_test, y_train, y_test, mlp)
```

MLPClassifier relies on an underlying Neural Network to perform the task of classification. A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs.

Different values were tried for alpha, max_iter and solver which resulted in the following accuracy results:

- **Training Accuracy – 37.65%**
- **Testing Accuracy – 37.40%**

AdaBoost Classifier

```
abc = AdaBoostClassifier(n_estimators=50, base_estimator= DecisionTreeClassifier(max_depth=8),learning_rate=0.01)
abc = abc.fit(X_train, y_train)
accuracy(X_train, X_test, y_train, y_test, abc)
```

Ada-boost classifier combines weak classifier algorithm to form strong classifier. A single algorithm may classify the objects poorly. But if we combine multiple classifiers with selection of training set at every iteration and assigning right amount of weight in final voting, we can have good accuracy score for overall classifier.

For AdaBoostClassifier, n_estimators used = 20, 50, 70, 100, 300, 500, 1000, max_depth = 3, 4, 5, 7, 8, 20, 15, 20 and learning_rate = 0.1, 0.3, 0.5, 0.01, 0.08, 0.003. The best accuracy results:

- **Training Accuracy – 73.54%**
- **Testing Accuracy – 61.29%**

Ensemble Method (Stacking)

```
# Decision Tree Classifier
model1 = DecisionTreeClassifier()
model1.fit(X_train_v, y_train_v)
val_pred1=model1.predict(X_val)
test_pred1=model1.predict(X_test)
val_pred1=pd.DataFrame(val_pred1)
test_pred1=pd.DataFrame(test_pred1)

# Random Forest Classifier
model2 = RandomForestClassifier()
model2.fit(X_train_v,y_train_v)
val_pred2=model2.predict(X_val)
test_pred2=model2.predict(X_test)
val_pred2=pd.DataFrame(val_pred2)
test_pred2=pd.DataFrame(test_pred2)
```

```
# MLP Classifier
model3 = mlp
model3.fit(X_train_v,y_train_v)
val_pred3=model3.predict(X_val)
test_pred3=model3.predict(X_test)
val_pred3=pd.DataFrame(val_pred3)
test_pred3=pd.DataFrame(test_pred3)

# k-Nearest Neighbor Classifier
model4 = KNeighborsClassifier()
model4.fit(X_train_v,y_train_v)
val_pred4=model4.predict(X_val)
test_pred4=model4.predict(X_test)
val_pred4=pd.DataFrame(val_pred4)
test_pred4=pd.DataFrame(test_pred4)
```

```
df_val=pd.concat([X_val, val_pred1.reindex_like(
df_test=pd.concat([X_test, test_pred1.reindex_li

df_val.dropna(axis='columns', inplace=True)
df_test.dropna(axis='columns', inplace=True)

# Getting the test accuracy score for Stacking
model = LogisticRegression()
model.fit(df_val,y_val)
model.score(df_test,y_test)
```

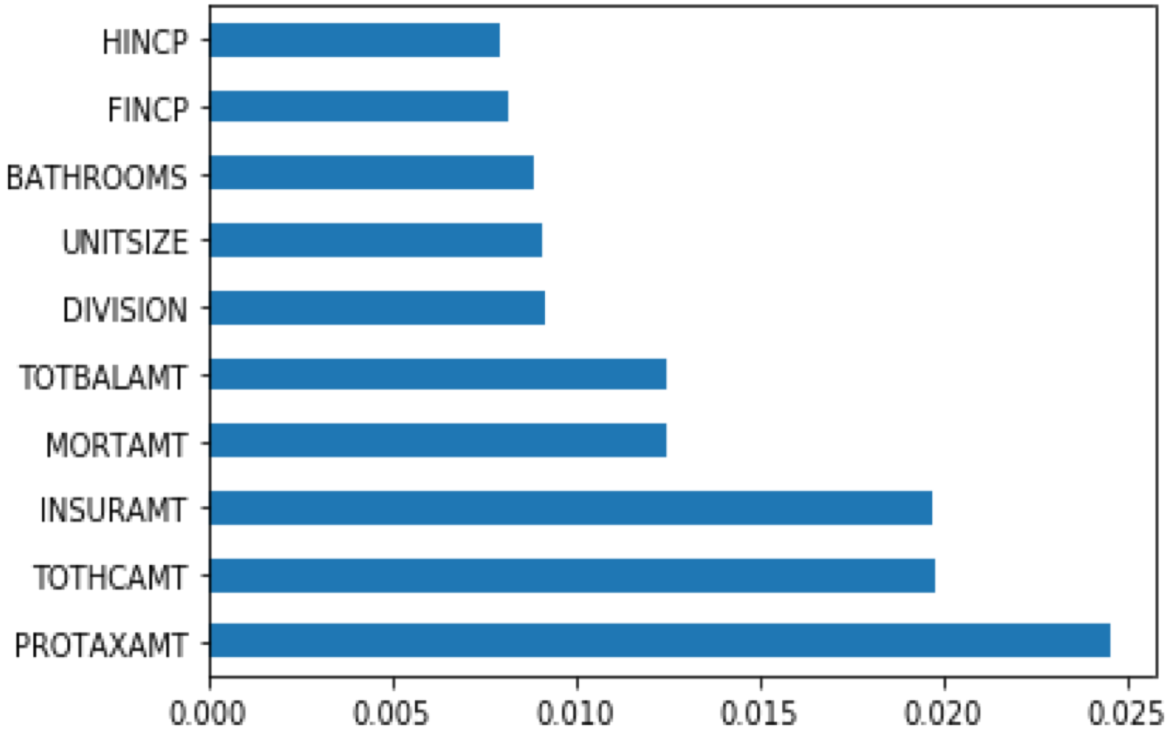
Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features.

With Stacking, the accuracy score were as below:

- **Testing Accuracy Score – 0.43436010267693437**

Feature Importance

Using Random Forest, the following bar graph was plotted representing the top 10 features based on their importance in determining the house price range.



These 10 features are as below:

PROTAXAMT - Monthly Property Tax Amount

TOTHCAMT - Monthly Total Housing Cost

INSURAMT - Monthly Homeowner or Renter Insurance Amount

MORTAMT - Monthly Total Mortgage Amount (all mortgages)

TOTBALAMT - Total Remaining Debt Across all Mortgages or Similar Debts for this Unit

DIVISION - Census Division

UNITSIZE - Unit Size (Square Feet)

BATHROOMS - Number of Bathrooms in Unit

FINCP - Family Income (Past 12 Months)

HINCP - Household Income (Past 12 Months)

Conclusion

- A range of different models were explored, from relatively simple to more complex ones, with the goal of best predicting home price ranges but also utilizing different data science skills in the process.
- For this particular problem, the algorithm with best accuracy value is AdaBoost Classifier with test accuracy score of 61.29% and therefore it can be considered as a good classifier algorithm for house price range prediction problem.
- The Decision Tree Classifier is close enough with 60.60% accuracy score.
- Each algorithm was tuned with different hyper-parameter values and finally the best results for each were retained, thereby attaining hyper-parameter optimization.

- Github Repository - *https://github.com/darpraithatha/House-Price-Range-Prediction*
- Google Drive - *https://rb.gy/jdzjde*

Individual Contribution

Each of the members had unique contributions towards the project. But there are areas where all three of us are working together as well.

- Darp Raithatha is responsible for preliminary research, gathering information, correlation determination, external materials to refer to, model evaluation and code testing.

- Likith Nandigam is responsible for cleaning the dataset, compiling the code, code testing, result extraction, and documentation.

- Monami Mukhopadhyay is responsible for code testing, result extraction, project compilation, documentation, gathering and implementing references.

stevens.edu

Thank you