Report of the Application of Machine Learning Project

Machine learning (ML) is the science of programming computers so they can learn from data (1). Machine learning algorithms build a mathematical model on a given data set also known as training data set and make predictions on unseen data set.
There are two general types of Machine Learning: Supervised and unsupervised ML.
In a supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels (1). In unsupervised learning, the training data is unlabeled. The algorithm tries to learn without supervision.

For my project, I obtained a data from the website, NYC Open Data. I wasn't sure what data to use for the application of machine learning, so I searched the given websites to find a data to make predictions. I found the data, "NYC Citywide Annualized Calendar Sales Update" which is provided by the Department of Finance. I chose this data set to apply machine learning to predict the properties' sale prices in NY City.

I uploaded the data in a jupyter notebook using the pandas pd.read.csv function. This function creates pandas Data Frame object containing all the data. This data set has 345 thousand rows and 29 columns. Each row has an Annual Sales Information for Properties Sold in New York City. The columns are *borough*, *neighborhood*, *building class category*, *tax class as of final roll*, *address*, *apartment number*, *zip code*, *residential units*, *commercial units*, *total units, land square feet, gross square feet, year built, tax class at time of sale, sale price, sale date, latitude, longitude, community board, council district, census tract, bin, bbl*, and *nta*.
"The column BIN (Building Identification Number) is a unique identifier for each building in the City" (3). "The BBL (Borough, Block, and Lot) is a unique identifier for each tax lot in the City" (3). "The Neighborhood Tabulation Area field indicates the New York City Neighborhood area where the building is located" (3).

Pre-Processing

Splitting The Data

Most of the machine learning tasks are about making predictions. This means that the algorithm needs to be trained well with enough data samples in order to perform in the same way on the given unseen new data. This is called *generalizing* the algorithm (1). Having a good performance on the training data is good, but it's not enough unless it also performs well on the new instances. In order to train the data and measure the performance on unseen data, I split my data as 80% for training and 20% for testing. Scikit-Learn provides some functions to split datasets into multiple subsets in various ways (1). To use scikit – learn's split function, first I import the function, *'train_test_split'* from '*sklearn model selection'* . This function takes the parameters such as predictor features, 'X', target variable, 'y' and a random generator seed.
The random generator seed prevents the algorithm to mix the test set samples with the training set in subsequent runs. Thus, the test set will never be seen by the algorithm before.

Note: I've realized that I did some exploration and visualization of the data before I split the data to a training and a testing set, which is wrong. I had to do that on the training dataset after splitting the data.

Cleaning The Data

The first step, before applying any ML algorithm to the data is exploring the data on the training set. This would be done by the ".info()" method. If a column in the data has some missing values, one of the options is to fill the missing values with either zero or with the median of the feature. If it has a lot of missing values, getting rid of that column is one of the options as well.

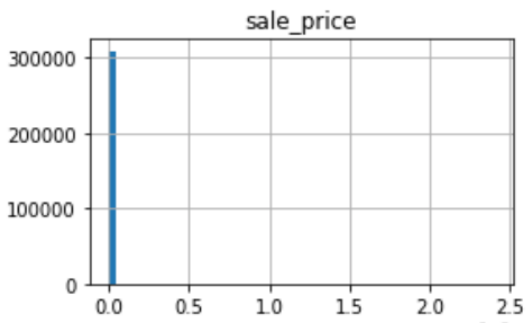
Using the .info() method, I found out that 10 of the 29 columns are object type (text or date) and the rest are numerical type, either integer or float. Since the column "easement" has zero non-null object and the "apartment number" has many null objects, I got rid of them. Also, I dropped the rows that have missing values to clean the data.

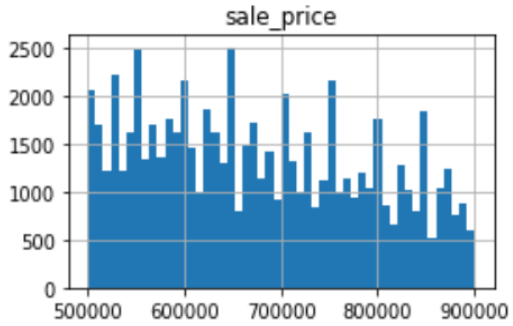
The statistical information of the data can be obtained by the .describe() method. The function, .describe() gives the mean, median, standard deviation, min value, max value, 25^{th} percentile, 50^{th} percentile, and 75^{th} percentile of the data. Besides, plotting the histogram of the numerical attributes of the training data would provide some exploration of the data.

So I got the statistical information of the data by the .desribe() function. The 'sale price' which is the target variable, has a minimum value of zero and a maximum value of m 2 billion and 300 thousands. I realized that there are many samples that have the sale price of zero. This happens because of changing the owner of the properties without any cost (i. e., from parents to kids). Since the sale price of zero would not be helpful to predict the sale prices of properties, I filtered them out.

When I plot the histogram of the data, the histogram of the 'sale price' looked like a vertical bar. First, I was skeptical and wasn't sure about the reason. Then, I realized that it was because of the samples which have zero for the sale price or it would be from the outliers as well. When I filtered out the zeros and the outliers by just keeping the data in a certain range, the histogram shifted to a better shape.

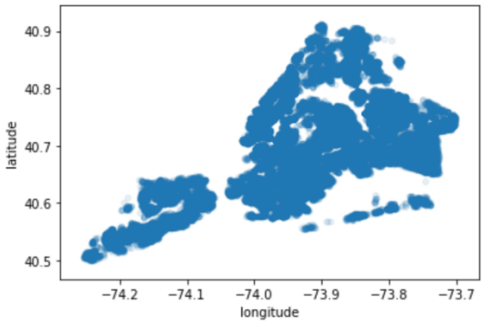
The histogram of the target variable, sale price before and after filtering out zeros and the outliers:





Next, I looked at the correlation matrix of the numerical attributes of the data with the target variable, sale price. The correlation matrix indicates that the 'total units' has the highest positive relationship with the sale price even though it's not strong. Whereas, the 'residential units' takes the second position.

I also, looked at the geographical distribution of the training data to get a deep insight of it.



The graph visualize the New York City, and it's difficult to make any conclusion based on this plot because it's too dense.

Handling Text and Categorical Attributes

Most of the Machine Learning algorithms work with numerical data, so we need to convert the columns which are text type to numerical. Scikit-Learn provides transformer functions such as '*LabelEncoder*' and '*OneHotEncoder*'. If we use the function, 'LabelEncoder()' it will transform the text data to a numerical data, however, there's one issue that the algorithm will assume that two nearby values are similar than the two distant values (1). To fix this issue, a common solution is to create one binary attribute (1). We can create a binary 2 dimensional array by using the Scikit-Learn's function, 'OneHotEncoder'.
I used the function, OneHotEncoder to convert some categorical text features to a numerical 2D array. For example, the feature 'tax_class_as_of_final_roll' which gives the present tax class of the properties is an object type and it has 10 unique values.

After using the 'OneHotEncoder' function, I obtained the following 2D array:

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Transformation Pipelines

To execute the transformations of the data in the right order, we use the Scikit-Learn's *Pipeline* class. Then, we can combine the transformed data by the numerical data by using the Scikit-Lea rn's 'FeatureUnion' class. After I combined the transformed and the numerical data, my training data had 26 columns.

Supervised Learning

One of the supervised learning tasks is to predict a target numeric value, such as the price of a house, with given a set of features (i. e., the community, the total rooms of the house, the year built in, etc.) called predictors. This task is called regression. To train the algorithm, many samples should be given, including both, predictors and labels.
First, we import the *LinearRegression* model from Scikit-Learn and instantiate it. Then, we train the model by passing the predictor features and the target variables from the training dataset.
After I trained the Linear Regression model, I looked at a few instances from the training dataset:
Predictions: [698606, 599325, 667851, 657984, 692337]
Labels: [710000, 550000, 660000, 577000, 665000]
Predictions are not accurate, but they're relatively close to the labels.

Evaluation Metrics

To evaluate the performance of the Regression algorithm, three evaluation metrics are commonly used:

1. Mean Absolute Error (MAE) is the mean of the absolute value of the errors. It is calculated as:

$$MAE = \frac{1}{n}\sum_{j=1}^{n} |y - \bar{y}|$$

   Where y is the actual output and y bar is the predicted output.

2. The Mean Square Error (MSE) is the mean of the squared errors and is calculated:

$$MSE = \frac{1}{n}\sum_{j=1}^{n} (y - \bar{y})\text{^}2$$

3. The Root Squared Mean Error (RMSE) is square root of the MSE and is calculated:

$$RMSE = \sqrt{\left(\frac{1}{n}\sum_{j=1}^{n} (y - \bar{y}\right)^2}$$

The result of RMSE metric on the training is 109,312. This result indicates that the features do not provide enough information to make good predictions. This also means that the model is underfitting. One of the ways to fix that is to select a more powerful model.
And the result of MAE on the training data is 92,969 which is the absolute average error. MAE does not indicate underperformance or overperformance. A small MAE suggests that the model is great at prediction while a large MAE suggests that the model may have problems in some areas (4).
Next, I applied a DecisionTreeRegressor which is a powerful model, capable of finding complex nonlinear relationships. The DecisionTreeRegressor algorithm breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with *decision nodes* and *leaf nodes* (5).
The RMSE of this model is 35,205 which is smaller than the RMSE of LinearRegression.
Better Evaluation Using Cross-Validation
One way to better evaluate the Decision Tree model is using Scikit-Learn's cross-validation feature. Cross_validation feature splits the training set into a smaller training set and a validation set, then train your models against the smaller training set and evaluate them against the validation set (1).
I applied the cross-validation metric with 10 cross validation folds on the DecsionTreeRegressor and on the LinearRegressor and I got a better result on the LinearRegressor.

Random Forest Regressor

Random Forest regressor trains many Decision Trees on random subsets of the features, and it gives the average of the predictions (1). "Building a model on top of many other models is called Ensemble Learning" (1).
The result of the RMSE on the Forest Regressor is 47,945. Which is higher than the Decision Tree Regressor.

Fine-Tune Your Model

To find the best model, we manipulate the hyperparameters until we find the best combination of hyperparameters. For this, we Scikit-Learn's GridSearchCV by passing various hyperparamete rs. It will evaluate all possible combinations of hyperparameters using cross-validation.
I tried 3X4 combination and 2X3 combination and got best parameters as {'max_features': 6, 'n _estimators' : 30}. The RMSE for this combination is 99,741.

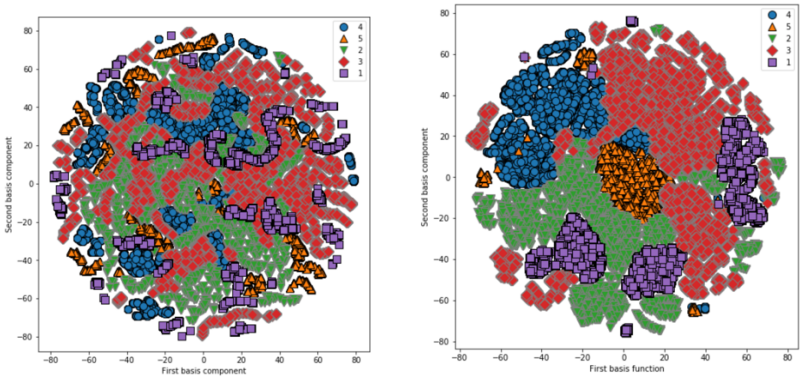
Note: Since 6 and 30 are the maximum values that were evaluated, I should've done more search to improve the score.
Next, I applied the RandomizedSearchCV. This class evaluates a given number of random combi nations by selecting a random value for each hyperparameter at every iteration (1). The RMSE of the combination of {'max_features': 3, 'n_estimators' : 150} is 98,879.

Evaluating the Model on the Test Set

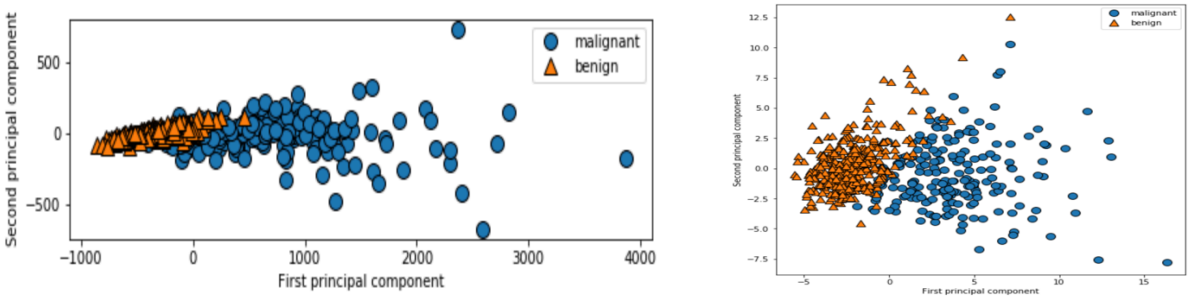
To evaluate the model on the test set, we run 'full_pipeline.transform' on the X_test set. Then we run the 'grid_search.best_estimator_' as a final model.
The RMSE of final mode on the test set is 98,339. Which is close to the training model result.

Unsupervised Learning

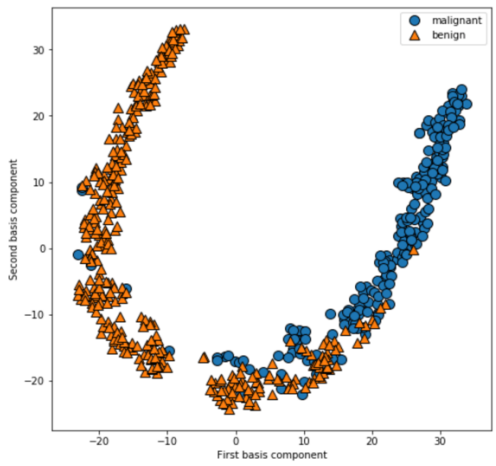
Most broadly used algorithm for unsupervised ML is Principal Component Analysis (PCA). PCA is the most popular dimensionality reduction algorithm, but it's also useful as a tool for visualization, and noise filtering. First, it identifies the closest plane to data points, then it projects the points onto the plane (1). PCA identifies the axis that holds for the largest amount of variance (information) in the training set (1). It also finds the second axis, orthogonal to the first axis, that accounts for the largest amount of remaining variance, and the third, the fourth and so on (1).
Scikit Learn's PCA class implements PCA using Singular Value Decomposition (SVD) and it also, Centers the data (1).
Explained_variance_ratio_variable indicates the proportion of the dataset's variance that lies along the axis of each principal component (1).
I applied the PCA on my data stating the target variable as 'borough' which the properties located in. First I ran the PCA with scaling the training data, with 2 components. But I didn't get any clustered data points. After I scaled the training data, I got the clusters, but they weren't separated nicely. The variance ratio of the first component is 0.3134 that means the first princip al component accounts 30% of the information, whereas, the second component holds 1.4 % of the variance. I applied t-distributed Stochastic Neighbor Embedding (t-SNE), a tool to visualize a high dimensional data on my unscaled and scaled data and I got mixed clusters of the data points.

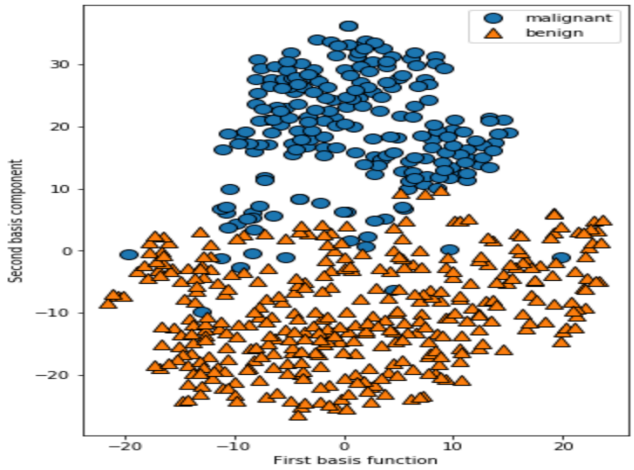


Next, I ran the PCA on the Breast Cancer data for the target names 'malignant' and 'benign'. First, I ran the PCA on unscaled data and then on the scaled data:



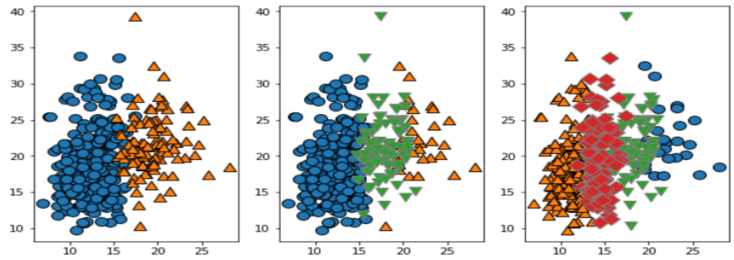
Also, I applied the t-SNE on the Breast Cancer on the unscaled and the scaled training data and I obtained the following visualizations:



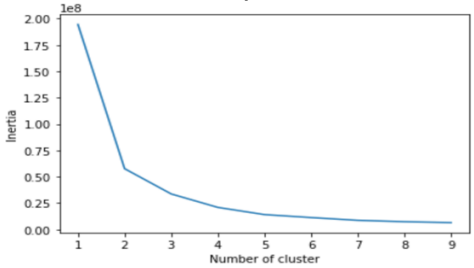


I've realized that the clusters on the unscaled data is dense, whereas the clusters on the scaled data are more separated on the area.
Explained variance ratio for the first component is 0.4427 and for the second component is 0.18 971. This means the first principal component accounts the 44% of the variance, whereas the se cond component holds 19% of the variance.
Using the Scikit Learn's Logistic Regression Model, I measured the accuracy of the model and got 99% of accuracy on the training data and 98% of accuracy on the testing set.

K-means Clustering
The *k*-means algorithm searches for a pre-determined number of clusters within an unlabeled multidimensional dataset (6). The optimal cluster requires to be:
The cluster center is the mean of all the points belonging to the cluster. And each point is closer to it's cluster center than to the other cluster centers (6).
The k-means clustering on Breast Cancer training data with different number of clusters:



To determine the optimal number of clusters for training data, I plotted the elbow graph:

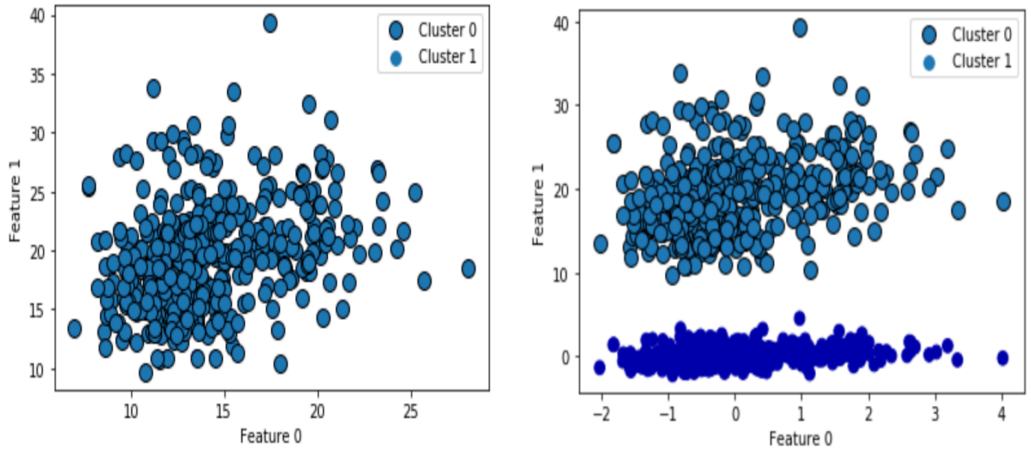


Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm used in ML. DBSCAN groups the points that are close to each other based on a distance measurement and a minimum number of points. It marks the points in a low-density regions as outliers. It requires two parameters:

Eps: specifies how much the points should be close to each other to be considered as a part of the cluster. If the distance between two points are less than or equal to eps, they're considered neighbors.

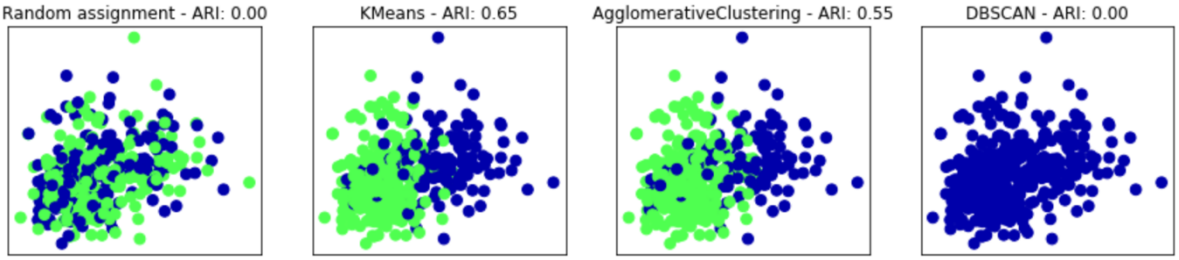
minPoints: minimum number of points to form a cluster. For example, if the minPoints is 5, at least there should be 5 points to make a cluster.

DBSCAN clustering on the unscaled and the scaled data:

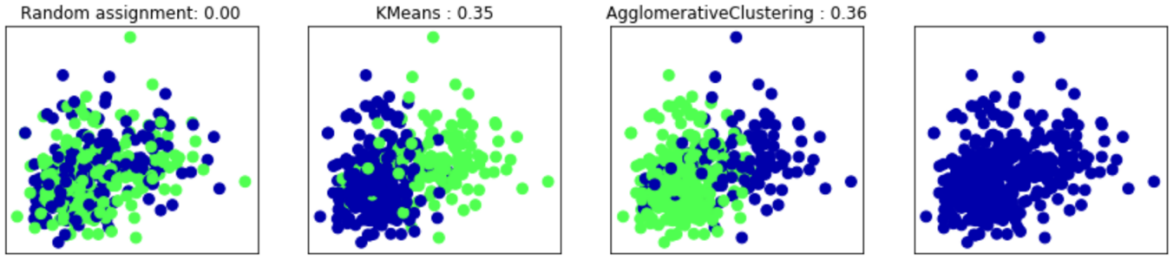


The Adjusted Rand Index (ARI) is used to measure the similarity of datapoints presents in the clusters i.e., how similar the instances that are present in the cluster. So, this measure should be high as possible. Otherwise, we can assume that the datapoints are randomly assigned in the clusters.

Lastly, I compared the clusters with adjusted rand index with the ground truth and obtained the following result:



Comparing clustering algorithms without ground truth:



In the first graph, the K-means clustering and the Agglomarative clustering have a high ARI value. That means the points in that clusters are similar to each other.

Summary

I uploaded the data and split the data to the training and testing set. I cleaned the training set, and converted the categorical variables to the numerical types using Scikit Learn's 'OneHotEncoder' class. After preparing the training data, I used the LinearRegressor model from Scikit Learn to predict the sale prices. To get the smallest RMSE, I tried various algorithm, then test the model on unseen data.
For the unsupervised learning, I applied the PCA on my data, but it didn't work well. I need to search up the reason that it didn't work well.
Overall, this is my first Machine Learning project. I've learned a lot with this project, but there's still a lot to learn. First, when I upload the data, I need to split the data to the training set and the testing set with Scikit Learn's 'train_test_split' function. Then, I should explore and clean the training data. I made a mistake in that part. To get a better result from the Linear Regression model, I would include more features by converting the rest of the text attributes to the numerical types. Also, I should try more various hyperparameters to get the best result. The other point is if the feature values have large deviation, scaling the data would help to get the optimal result as well.

References:

(1).Geron, Aurelien, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*, O'Relly Media, Inc., US, 2017
(2). Müller, Andreas C. & Guido,Sarah, *Introduction to Machine learning with Python*, O'Relly Media, Inc., US, 2016
(3). https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a
(4). https://www.dataquest.io/blog/understanding-regression-error-metrics/
(5). https://saedsayad.com/decision_tree_reg.htm
(6). https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html