Machine Learning – Assignment 3: Regression

Luke Gibbons – 4BCT – 15553883

ML Package Choice:

I have decided to use the Python package: scikit-learn. This is the same package that I used for the classification task in assignment 1, and so I have some experience using it. It is an easy to use library and comes with many algorithms that can be used out of the box for both classification and regression tasks. On top of this, it has many tutorials and documentation scattered across the web which make it easy to find help when problems occur during model building.

Scikit-learn also provides a number of classes outside of ML algorithms that could prove useful for this assignment, notably metrics which can be used for evaluation, and train_test_split which will be helpful for this task due to all the data being contained in one file.

Preparing the Data:

Preparing the beer ratings file for input into the package is a very straightforward process.

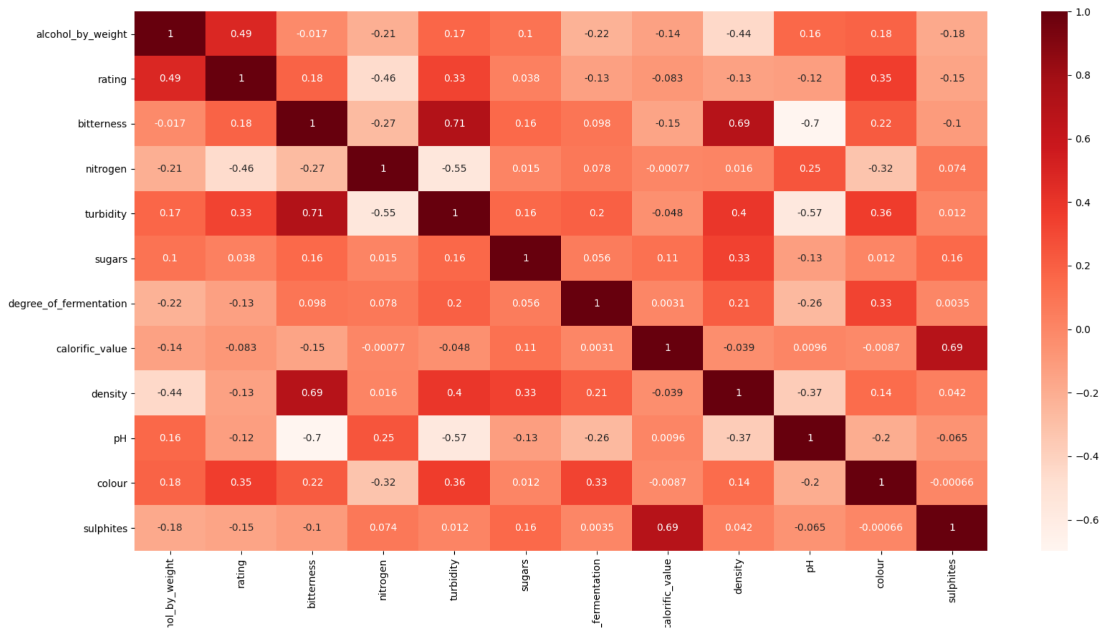
First, the data was loaded which required the use of the Python package Pandas. This allows the conversion of a text file into a data frame (a two-dimensional labelled data structure).

The headings (*alcohol_by_weight*, *rating*, *bitterness*, etc.) were then applied to the data frame. This will make it easier to reference a certain column when separating the data into features and labels.

Feature Selection and Dataset Dividing:

It is important that we avoid both overfitting and underfitting in our model. Doing so requires taking two precautions: removal of unnecessary features, and training with more data.

To find unnecessary input features we will use a filtering method. This requires filtering the features and using only a subset of those relevant. To accomplish this requires the use of a Pearson correlation heatmap, which will allow us to see the correlation of independent variables with the output variable *rating*. The generated heatmap can be seen below:



A value closer to 0 in the above heatmap implies a weaker correlation while one closer to -1 or +1 implies a stronger one (both negative & positive, respectively). Using a benchmark of +/- 0.35 we can determine that the best features to use (highest correlation to *rating*) are *alcohol_by_weight* (0.49), *nitrogen* (-0.46), and *colour* (0.35).

As I am using a Linear Regression (as will be seen in the next section) the independent variables must be uncorrelated to one another, as is an assumption of the algorithm. Looking again at the heatmap, it may be determined that each feature is uncorrelated to the others as they don't pass the +/- 0.35 benchmark.

Now that we have dropped irrelevant features and are left with only relevant ones we can proceed in splitting the dataset into training and test data. The scikit-learn method, train_test_split, is used for this and a split of 70% training / 30% test split is chosen. This gives the plenty of data to train with which will, hopefully, help in avoiding both overfitting and underfitting of the model.

The Algorithms:

The two algorithms chosen for the assignment are: Linear Regression, and K Nearest Neighbours (k-NN).

Linear Regression

Linearity refers to a linear relationship between two or more variables, which, when drawn in a two-dimensional space, results in a straight line.

So, in a scenario whereby a relationship is to be determined between the number of hours a student studies and the percentage of marks a student scores in an exam, we can plot the independent variables (hours) on the x-axis and the dependent variables (percent) on the y-axis and use linear regression. The algorithm will find the straight line with optimal intercept and slope values that best splits the data points.

The above case, however, only deals in two dimensions. If more variables exists then multiple linear regression is used, as is in our beer dataset where the dependent variable (*rating*) is dependent upon several independent variables (*alcohol_by_weight*, *etc.*). A regression model in this case may be represented as:

$$y = b_0 + m_1 b_1 + m_2 b_2 + m_3 b_3 + \cdots + m_n b_n$$

Where *y* is the hyperplane, *b* the intercept, and *m* the slope.

<u>K Nearest Neighbours (k-NN)</u>

The k-nearest neighbors algorithm predicts a label by the assumption that similar data exists in close proximity. It uses methods to find a number of training samples closest in distance to the new data point, it can then predict a label based on these 'neighbors' (by taking their average). A number of different techniques may be used to calculate the distance between points, the most popular being Euclidean distance (also known as straight line distance).

The value *k* represents the number of training samples (nearest neighbors) that are taken into consideration. The value of *k* is highly data dependent as a larger value will suppress the effects of noise but will also make the classification boundaries less evident.

Developing the Models:

<u>Development</u>

Once both train and test datasets have been split into features and labels (x and y), initializing the algorithms with scikit-learn is easy.

The algorithms are first initialized and their parameters are entered. From here, a simple *fit* method, using the training data, is called for each algorithm. The models are then developed.

<u>Parameter options</u>

Decisions on the parameters were determined through researching the various options each presented. A run through for each algorithm can be found below:

Logistic Regression Options		
Parameter	Description	Decision
fit_intercept	Bool - Whether to calculate the intercept of the model.	True – the beer dataset is not centered so we will require the intercept to be used in calculations
normalize	Bool – if True, the regressors X will be normalized before regression by subtracting the	True – normalization is required when features have different ranges. While our features don't differ too much,

	mean and dividing by the l2-norm.	there is slight variation between them (*alcohol_weight* = 4-5, *nitrogen* = 0-1, *color* = 5-16), so we will normalize them to be safe.
copy_X	Bool - If True, X will be copied, else, it may be overwritten.	False – there is no need to copy X as it's not used anywhere else.
n_jobs	Int – the number of jobs to use for the computation	1 – no need for a large number here due to using a small dataset.

K-NN Options		
Parameter	Description	Decision
n_neighbours	Int – number of neighbors to use	5 – this may require a trial and error approach to find the best value, though 5 is a good start as it is odd, and should avoid overfitting / underfitting.
weights	{'uniform', 'distance'} – the weight function used in prediction. Uniform weights (all equal) or distance (closer neighbours have greater influence)	Uniform – this is the most common method used in k-NN, so we will initialize it with this
algorithm	{'auto', 'ball_tree', 'kd_tree', 'brute} – the algorithm used to compute the nearest neighbours.	Auto – this is the best option as it will choose the best option for us depending on the beer dataset values.
leaf_size	Int – leaf size passed to BallTree or KDTree.	Default (30) – this can remain default as it only affects speed of construction and required memory to store the tree.
p	Int – the power parameter for the Minkowski metric. If p=1, equivalent to Manhattan distance, if p=2, Euclidean distance.	2 – Euclidean may be the best option in this case. This is due to the fact that, we have a low number of dimensions (from performing feature selection). Manhattan is preferable is high dimensional data
metric	String – the distance metric to use for the tree.	Default (Minkowski) – this is the best option as we are looking to use the Euclidean distance.
n_jobs	Int – the number of jobs to use for the computation	1 – no need for a large number here due to using a small dataset.

Model Evaluation:

The models are evaluated using the following three metrics:

1. Mean Absolute Error (MAE): the mean of the absolute value of the errors.

2. Mean Squared Error (MSE): the mean of the squared errors.

3. Root Mean Squared Error (RMSE): the square root of the mean of the squared errors.

The result were as follows (average after 5 runs, each having a different train-test split):

	Linear Regression	**k-NN**
MAE	6.155	5.996
MSE	62.92	64.038
RMSE	7.926	7.997

From the results we can see that both algorithms did a decent job overall, having both a low MAE and RMSE. They also produced extremely similar results. The similarity in results may come down to the feature selection, as with a higher dimensionality K-NN may not have performed as well as the Linear Regression algorithm.

References

1. Aggarwal C. et al. 2001, *On the Surprising Behavior of Distance Metrics in High Dimensional Space*, ICDT 2001, page 420-434, viewed 20 December 2020, <https://bib.dbvis.de/uploadedFiles/155.pdf>

2. Lakshmanan S. 2019, *How, When, and Why Should You Normalize / Standardize / Rescale Your Data?*, Medium, viewed 20 December 2020, <https://medium.com/towards-artificial-intelligence/how-when-and-why-should-you-normalize-standardize-rescale-your-data-3f083def38ff>

3. Miller M. 2019, *The Basics: KNN for classification and regression*, towards data science, viewed 20 December 2020, <https://towardsdatascience.com/the-basics-knn-for-classification-and-regression-c1e8a6c955>

4. Robinson S. 2017, *Linear Regression in Python with Scikit-Learn*, Stack Abuse, viewed 20 December 2020, <https://stackabuse.com/linear-regression-in-python-with-scikit-learn/>

5. Scikit-learn developers 2020, *sklearn.linear_model.LinearRegression*, Scikit-learn, viewed 17 December 2020, <https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html>

6. Scikit-learn developers 2020, *sklearn.neighbors.KNeighborsRegressor*, Scikit-learn, viewed 17 December 2020, <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html>

7. Scikit-learn developers 2020, *sklearn.model_selection.train_test_split*, Scikit-learn, viewed 18 December 2020, <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>

8. Shetye A. 2019, *Feature Selection with sklearn and Pandas*, towards data science, viewed 20 December 2020, https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b