K-NEAREST NEIGHBOR CLASSIFICATION

BRENDAN WHITAKER

Abstract. We analyze the dataset `winequality-red` which contains several categories of quantitative information on red wines. The dataset was sourced from the UC Irvine Machine Learning Repository hosted by the Center for Machine Learning and Intelligent Systems. We explore the data and determine if we can draw any useful conclusions from correlation and summary statistics. We discuss several data transformations required to prepare the data for training, and evaluate the finished model against a pre-built implementation of the kNN classifier algorithm.

Contents

1. Introduction and preprocessing

We give some preliminary information about the dataset. It contains metadata for 1599 wines, possibly batches of wines. There are 12 column headers with information concerning the acidity, residual sugar, several other physical/chemical/gastronomical properties, and a real-valued "quality" variable which ranges from 0 to 10. The data was imported and analyzed using `python 2.7` in the Jupyter Notebook web application. The `NumPy`, `Matplotlib`, and `pandas` python libraries were imported in order to aid in data exploration, cleaning, and predictive modeling. the data was imported in a `pandas` dataframe, a structure similar to an Excel workbook.

We perform some simple data preprocessing by removing the quantitative "quality" metric and instead adding a column for "high" and "low" quality, where we say a wine has "high" quality if its quality measure is > 5, and label it "low" otherwise. Our code to do this is given by:

```
df_binary = df.copy()
df_binary['binary_quality'] = 0
```

Date: SP18.

```
for i in range(0,1599):
    if df['quality'][i] <= 5:
        df_binary.loc[i,'binary_quality'] = "low"
    else:
        df_binary.loc[i,'binary_quality'] = "high"

df_clean = df_binary.drop(columns=['quality'])
```

To get a simple summary of the data in our `df_clean` dataframe, we use the `describe()` function to obtain some simple summary statistics:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	1599	1599	1599	1599	1599	1599	1599	1599	1599	1599	1599
mean	8.319	0.527	0.270	2.538	0.087	15.874	46.467	0.996	3.311	0.658	10.422
std	1.741	0.179	0.194	1.409	0.047	10.460	32.895	0.001	0.154	0.169	1.065
min	4.6	0.12	0	0.9	0.012	1	6	0.99007	2.74	0.33	8.4
25%	7.1	0.39	0.09	1.9	0.07	7	22	0.9956	3.21	0.55	9.5
50%	7.9	0.52	0.26	2.2	0.079	14	38	0.99675	3.31	0.62	10.2
75%	9.2	0.64	0.42	2.6	0.09	21	62	0.997835	3.4	0.73	11.1
max	15.9	1.58	1	15.5	0.611	72	289	1.00369	4.01	2	14.9

TABLE 1. `df.describe()`

Just from this basic summary we already have some meaningful information available about these wines. For example, from the pH column, we know that all the wines have a pH below 4.01, and so all wines are nontrivially acidic, since any pH below 7, the pH of pure water, is considered acidic. We can also see that all the wines have a density extremely close to 1, since the mean is 0.996, and the standard deviation is only 0.001. This makes sense because wines are mostly water, and so we can infer that the density is likely measured in g/ml^3 even though we have no preliminary information about this, since the density of water in these units is 1. We can also see from the means of the SO_2 columns that on average, about $\frac{1}{3}$ of the SO_2 in wine is free.

We see no blatantly obvious evidence of missing data, since the count for each column is 1599, and there are no significantly "illogical" values in the quantitative data. Nor do we see any obvious outliers, though it is difficult to make an accurate judgment of this, since we don't have information on the units of all the columns. For this reason, we will refrain from omitting any rows of data. We also note here that the added column is absent from Table 1, since we changed its values to text ("high" or "low" quality), and so it's no longer quantitative, and so the `describe()` function doesn't display it.

Now we discuss basic correlation between columns. We use the following code snippet to compute the Pearson correlation coefficients between each pairwise combination of the columns in `df_clean` which are ratio variables:

```
df_clean.corr(method='pearson', min_periods=1)
```

We present scatter plots for the pairs of variables with the highest correlation coefficients, which were fixed acidity and citric acid (0.672), alcohol and density (-0.496), free SO_2 and total SO_2 (0.668), fixed and density (0.668), fixed acidity and pH (-0.683), and citric acid and pH (-0.542):

Note that many of these make sense given a bit of exterior knowledge. For example, since as mentioned above, lower pH values correspond to a higher acidity, it makes sense that we see a negative correlation of pH with fixed acidity and citric acid. It also makes sense that citric acid has a positive correlation with fixed acidity. We observe an interesting shape in the scatter plot of free SO_2 with total SO_2. There are no points below the line $y = x$, which is logical since there must be at least as much total SO_2 as there is free SO_2. The positive correlation between fixed acidity and density could be explained by the fact that adding acidic solute to water in the process of making wine would raise the density of the liquid. The negative correlation between alcohol and density doesn't seem to have an obvious explanation, but it make sense in context because alcohol is quite basic, and thus higher levels of alcohol would raise the pH (lowering acidity), which agrees with the positive correlation we see between acidity and density. We consider omitting some of the columns which correlate highly with other variables, in order to avoid double-weighting our model. However, since none of our variables have especially high correlation coefficients (we arbitrarily chose ≥ 0.75 as our threshold), we will not omit any.

In order to implement the k-nearest neighbors classification algorithm, we'll need a good proximity measure for the points in our space. We aim to predict the class wine quality from the remaining classes, of which there are 11. We note that each of these is a ratio variable, so we choose cosine similarity on row vectors in $\mathbb{R}^{11}$. This is computed using the equation:

$$S = \frac{A \cdot B}{||A||||B||}$$

where S is the similarity measure, which will range from 0 to 1 since all our values are positive. A, B are the vectors in $\mathbb{R}^{11}$.

Now we discuss splitting the dataset into a training set and a testing set for our kNN model. We do this by looping over each row in the `DataFrame`, and generating a random integer between 1 and 100, sending the current row to the training datatset if it is ≤ 75, and to the test set otherwise. This partitions around 25% of our dataset for testing, and leaves the rest for training.

Altogether, we made a few minor transformations of the data in order to prepare for training the model. We removed a continuous ratio variable

and replaced it with a binary classifier, and we discussed our decision not to remove any rows or columns. We also did a cursory check of the summary statistics for missing values finding no evidence of such issues. Finally, we partitioned our data into training and test sets which are representative of the population via random sampling.

Our implementation for this partition is as follows:

```
#training data
df1 = df_clean.copy()
#test data
df2 = df_clean.copy()
partition_array = []
trainingCard = 0
testCard = 0
for i in range(0,1599):
    prob = random.randint(1,100)
    if prob <= 75:
        partition_array.append(0)
        df1.loc[trainingCard] = df_clean.loc[i]
        trainingCard += 1
    else:
        partition_array.append(1)
        df2.loc[testCard] = df_clean.loc[i]
        testCard += 1
dfTrain = df1[:trainingCard]
dfTest = df2[:testCard]
```

2. Implementation and testing of the model

We implemented the kNN algorithm as outlined in Algorithm 5.2 in the slides from class. We looped over each element in the testing dataset, and then computed the cosine similarity of this row vector with each of the training rows. Then we sorted these in descending order to yield the k pairs with the highest cosine similarity. We then counted the number of "high" vs "low" classes in these from the training set, and set the max of these two numbers as the predicted class. We then found the posterior probability of this class by dividing the number of instances of this class in our k best matches by k. We set the predicted class, the actual class, parsed from the testing DataFrame, and the posterior probability as a row in our results DataFrame.

A notable difficulty encountered is the running time of the algorithm, which was over 10 minutes per run, due to the large number of rows in the training and test set.

3. Evaluation of the model

We give the confusion matrix for $k = 11$:

Table 2. k = 11

		Predicted Class	
		High	Low
Actual Class	High	196	15
	Low	45	160

For $k = 9$:

Table 3. k = 9

		Predicted Class	
		High	Low
Actual Class	High	182	34
	Low	67	133

And for $k = 7$:

Table 4. k = 7

		Predicted Class	
		High	Low
Actual Class	High	170	24
	Low	36	186

So we had 416 total test data points. The error rates were: 0.144, 0.243, and 0.144 respectively for each of the k values. We choose $k = 7$ and compute some more measures of performance. For this k value, the TP rate is 0.876, the TN rate is 0.838. The FP rate is 0.162, and the FN rate is 0.124. The recall is 0.88, the precision is 0.83. The f-measure is 0.85.

Unfortunately, the code to graph the ROC curve couldn't be made to work within the allotted time to finish the assignment, but we expect to see a non-smooth curve with negative concavity which remains above the line $y = x$ where FPR is on the x axis and TPR is on the y axis.

We used the `sklearn.neighbors` pre-built implementation of k-nearest neighbors to compare our implementation to something we know will work correctly. The data tables for all the measures of performance across all

values of k were quite large, so we omit them and simply discuss the differences in terms of percent between our implementation and the sklearn one. We say an average of 4.3% deviation across all performance measures of our model to theirs, and since this is rather small we conclude that it might be caused solely by sampling variance. So we can say that our model works about as well as the sklearn one.