Assessing the accuracy of the existing classifier model - QuestionMark

The reliability of the machine learning classifying model implemented by QuestionMark is measured in a manner that has proven to be incorrect. This blog update discusses this problem, and the solution.

Problem
Improving the classifying algorithm is important to QuestionMark because it limits the amount of human hours that is needed to check the outcomes. Improving the algorithm can only be done when we know what we are improving from, otherwise setting goals would not be realistic. The reliability of the algorithm is measured with macro recall, as this suits the overall aims of QuestionMark best. The micro recall will also be provided in this blog post as it provides reference. Without any improvements to the manner in which recall is measured, the values predicted on the validation data are the following:

```
macro recall score:  0.77511104049
micro recall score:  0.859539532393
```

The way in which these values are found will be discussed first, followed by the way in which this is improved. The first step to finding these values is extracting features from all product nuts, which are the first ingredients, names and brands of product nuts. These features are normalised, which entails removing stop words, removing all punctuation, removing capital letters where this is appropriate and including distinguishing features for ingredients, brands and names. After this is done all duplicates are removed - this based on the features we extracted up to this point - and only those product nuts that are associated with a usage that appears in at least 3 products are included. Once this is done, the strings with features are converted to a matrix with a method called CountVectorise. An import aspect here is that the binary statement is set to True, as this makes sure the counts are not included, only presence is measured. This matrix is our X, and our Y are the usages associated with the vectors in X. These vectors all present a product nut. X can also be though of as the data, and Y the target. X and Y are then split into two groups, one group with 80 percent of that data, and one group with 20 percent. The 80 percent group will be our train data, and the 20 percent will be the validation data. This is done to ensure these predictions are accurate, and no mistakes influence the outcome. The train data is then cross validated in the following manner.

```
clf = LinearSVC(random_state = 2, verbose = 1)
scores = cross_val_score(clf, X_train, Y_train, cv=5,
scoring='recall_macro')
```

The macro recall score that is returned is 0.77 (+/- 0.02). Apart from the problems relating to overfitting, there is one problem with the manner in which sklearn computes recall, which is that if there is no support for a class, the class is still part of the calculation for the macro value of recall. This means 0s are added when they should not be. I resolved this by calculating the average of the recall belonging to each classes, if the support for that class was at least 1. This changed the macro recall score on the validation set to 0.80. This relates only to the scores predicted on the validation set, but I believe this to be more accurate. The micro recall score does not change as it measure each product nut, not each class.
So the update values are:

```
macro recall score:  0.788133086876
micro recall score:  0.859539532393
```

The overall problem with this is that there is a lot of similar data in the products nut dataset. This is problem addressed in http://developers.thequestionmark.org/project/2017-usage_classification_feature_reduction#a-proper-measure-of-success.

Solution

A solution to this problem is introducing 'bubbles' around all the product nuts that are associated with one product. These bubbles make it possible that all the members of one product would either be in the train data or in the test data during cross validation, but not in both. These bubbles are implemented in the next code by a method called LeaveOneLabelOut. This means our classifier looks the following way.

```
cv = LeaveOneLabelOut(labels_TRAIN)

clf = LinearSVC(random_state = 2, verbose = 1)
scores = cross_val_score(clf, X_TRAIN, Y_TRAIN, cv=cv,
scoring='recall_macro')
```

The important added features are the labels. The labels are the product_id's that are presented in the products database. To obtain these, and also to check the amount of products a usage is related to, a new dataset needs to be constructed. This database is not so different from the data in product nuts, only that the usage and product_id are added as new keys in the dictionary related to each product nut. To then implement these labels in our classifier, all the product_ids are set to be a unique label. This means there are 51416 unique labels. Because LeaveOneLabelOut preforms cross validation with as many folds as there are labels, this is not feasible. The amount of folds we want in the cross validation is 5, just like the code without bubbles. To achieve this all unique labels are set to either 1 2 3 4 5 or 6. Six because we need one section of the data to be our validation set. The 5 sections are trained on, and return the following macro recall score.

```
macro recall score: 0.51 (+/- 0.01)
```

On the validation set sklearn returns 0.55, but as argued above, I believe the macro score to not be accurate, and the updated version (removed all classes with no support) is:

```
macro recall score:  0.613273809524
micro recall score:  0.728652300081
```

Conclusion

In conclusion the macro recall before bubbles is 0.80 and with the bubbles it is 0.61. This difference is quite substantial, but with an accurate measurement method improvements will be much easier as we can see what improves the model, and what does not.