Machine Learning: Assignment 1

Name: Marcel Aguilar Garcia Student Id: 20235620

October 25, 2020

1 Question 1: Open-source machine learning package

I have decided to use **Scikit-learn** for this assignment. Scikit-learn is a free software machine learning library for the Python programming language. As the task is not complex and the dataset is relatively small, I was searching for a package that is accessible and simple. Scikit-learn provides easy access to different classification algorithms, K-Nearest Neighbors being one of them. Additionally, Scikit-learn provides other tools such as sklearn.preprocessing which allows to easily process the data before training the model. [3]

Finally, Python is one of the programming languages supported by The Jupyter Notebook. The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. I find Jupyter Notebook to be very practical for academic purposes. [1]

2 Question 2: Data Preparation

For convenience, I have reordered the features and moved the target, 'style', to the last column. Let's take a look to the first entries of the dataset:

```
In [4]: train.head()
```

Out[4]:

	calorific_value	nitrogen	turbidity	alcohol	sugars	bitterness	beer_id	colour	degree_of_fermentation	style
0	45.305310	0.459548	1.917273	4.227692	16.67	12.568947	167	11.04	62.178571	ale
1	43.889381	0.548977	3.186364	4.289231	16.73	14.974000	128	13.44	63.032857	ale
2	41.588496	0.542847	1.568182	4.344615	16.48	11.848789	88	14.04	63.468571	ale
3	44.553097	0.480301	1.871818	4.424615	18.59	13.879632	147	12.48	63.531429	ale
4	41.013274	0.441860	2.345455	4.264615	16.35	12.186053	74	12.12	63.747143	ale

First, I have verified that the dataset does not contain NaNs as this could lead to issues when training the model. As well, I have checked the data types to see which are numeric and which may need to be encoded.

```
In [6]: train.isnull().values.any()
Out[6]: False
```

```
In [8]: train.dtypes
Out[8]: calorific_value      float64
        nitrogen             float64
        turbidity            float64
        alcohol              float64
        sugars               float64
        bitterness           float64
        beer_id                int64
        colour               float64
        degree_of_fermentation float64
        style                 object
        dtype: object
```

The target, 'style', needs to be encoded in order to be processed by the algorithm. It is a nominal category as there is no logical ordering within its values (ale, larger and stout). I have decided to use a simple encoding by mapping each value to a different integer.

```
In [16]: style_encod = {'ale':0,'lager':1,'stout':2}
         train['style'] = train['style'].apply(lambda x: style_encod[x])
         test['style'] = test['style'].apply(lambda x: style_encod[x])
```

As the attribute 'beer_id' is a unique identifier, it does not provide any useful information. As well it can be seen that it's highly correlated to 'calorific_value', therefore it is duplicating information that we have. I have decided to remove this column. Finally, I have kept the target in a different variable and I have removed it from the dataset as the algorithm will handle the target separately.

```
In [77]: target_train = train['style']
         target_test = test['style']
         train.drop(['style','beer_id'],axis=1,inplace = True)
         test.drop(['style','beer_id'],axis=1,inplace = True)
         columns.remove('beer_id')
         columns.remove('style')
```

Now, we have both, train and test sets, with numeric attributes (float64) and ready to be trained by a model.

Note that I have done the same data pre-processing to both train and test sets.

3 Question 3: Classification Algorithms

It is important to note that Scikit-learn uses optimized versions of ID3, C4.5 and kNN algorithms.

The Scikit-learn version of Decision Tree Classifier uses a Classification and Regression Tree algorithm (CART) and not ID3 or C4.5. [4]

On the other hand, the package of Scikit-learn for kNN allows you to use between three different algorithms: ball_tree, kd_tree and brute. I believe that brute-force search is the most similar to the original one. Therefore, I have decided to use it for this assignment. [5]

As seen in the lectures, kNN finds a predefined number of training samples closest in distance to the new point, and predict the label from these. By default, Scikit-learn uses Minkowski distance with $p = 2$, this is equivalent to Euclidean distance. If the goal of this assignment was to have a model with high accuracy, I would be using p as a hyper-parameter to get better results.

I have chosen Random Forest Classifier as the second classification algorithm. Random Forest is an ensemble learning method for classification or regression. A Random Forest Classifier fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. As this is a small dataset it would probably be sufficient to use a single Decision Tree, but a Random Forest Classifier should be able to provide good results as well. [7]

4 Question 4: Training models

In a usual scenario, I would use a train, a validation and a test set. Following the instructions of this exercise, I decided not to use the validation set. Therefore, I would be using the whole train set in order to train my model.

A Random Forest Classifier can be trained with default parameters using Scikit-learn as seen below:

```
In [30]: from sklearn.ensemble import RandomForestClassifier
         rfc = RandomForestClassifier()
         rfc.fit(train, target_train)
```

As kNN is a distance-based algorithm, it is always a good idea to scale the data. I have used StandardScaler from scikit-learn.preprocessing which standardize features by removing the mean and scaling to unit variance. [6]

```
In [30]: from sklearn.preprocessing import StandardScaler
         scaler = StandardScaler()
         train = scaler.fit_transform(train)
         test = scaler.transform(test)
```

Now, we can finally train the classifier as seen below. For this exercise, the number of neighbours has been set to 3:

```
In [21]: from sklearn.neighbors import KNeighborsClassifier
         classifier_knn = KNeighborsClassifier(n_neighbors=3,algorithm = 'brute')
         classifier_knn.fit(train, target_train)
```

Using more than one performance metric can help to understand better the results of the model. A confusion matrix would help us understand which are the misclassified examples. For this case, as this is not an imbalanced dataset, it should be sufficient having a high accuracy:

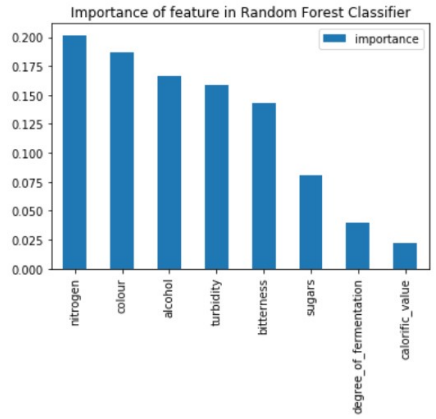
```
In [51]: accuracy_train_rfc = round(accuracy_score(target_train, rfc.predict(train)),3)
         accuracy_test_rfc  = round(accuracy_score(target_test, rfc.predict(test)),3)
         accuracy_train_knn = round(accuracy_score(target_train, classifier_knn.predict(train)),3)
         accuracy_test_knn  = round(accuracy_score(target_test, classifier_knn.predict(test)),3)
         print(f'Accuracy of kNN in train set is {accuracy_train_knn } and in test set is  {accuracy_test_knn}')
         print(f'Accuracy of rfc in train set is {accuracy_train_rfc } and in test set is  {accuracy_test_rfc}')

         Accuracy of kNN in train set is 0.984 and in test set is  0.967
         Accuracy of rfc in train set is 1.0 and in test set is  1.0
```

While visualizing all the trees from a random forest may give us an image that is too large to fit in this exercise (there are 100 trees when using default parameters), Scikit-learn allows you to check the importance that the model gives to the features of the dataset. The same way that a decision tree has features that allow the algorithm to gain more information, a random forest classifier can extrapolate this idea to each of the trees that are used. A bar plot can help visualizing the importance per feature:

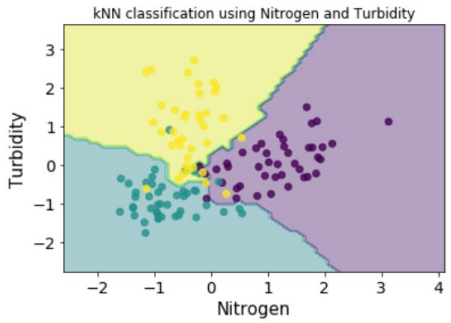
```
In [125]: df = pd.DataFrame(rfc.feature_importances_,index=columns,columns = ['importance']).sort_values(by=['importance'],ascending=False)
          df.plot(title = 'Importance of feature in Random Forest Classifier', kind = 'bar')

Out[125]: <matplotlib.axes._subplots.AxesSubplot at 0x170d7fe28c8>
```



For a dataset with n features, kNN will use an n-Dimensional Euclidean distance. Therefore, there is no easy way to visualize the classification when the number of features is greater than 3. For this exercise, I have selected Nitrogen and Turibidity to plot the classification of a kNN classifier using the train dataset with just these two features:

```
In [174]: plt.figure()
          plt.title("kNN classification using Nitrogen and Turbidity")
          plot_decision_boundaries(X,y,1,2,KNeighborsClassifier,n_neighbors=3)
          plt.show()
```



I have adjusted the code from [2] to this train dataset.

5 Question 5: Final results

For the final results, and basing the performance of the models on the accuracy of each one with the train and test sets, it can be seen that they perform good and have very similar results:

Model	Train Accuracy	Test Accuracy
Random Forest	1	1
kNN	1	0.967

Both models are able to classify perfectly all examples from the train set. However, kNN misclassifies some of the examples from the test set. This could be from the fact that we have selected 3 neighbours for the kNN classifier and a small value of k could lead to overfitting or just the fact that our data is not enough for the model. While random forest classifier could as well overfit, it seemed to work perfect in our example. In general, increasing the number of trees of the forest should decrease the overfitting but in this case, there was no need.

References

[1] *Jupyter*. URL: `https://jupyter.org/`.

[2] *Plot the decision boundaries of a VotingClassifier*. URL: `https://scikit-learn.org/stable/auto_examples/ensemble/plot_voting_decision_regions.html`.

[3] *Scikit-Learn Machine Learning in Python*. URL: `https://scikit-learn.org/stable/`.

[4] *Scikit-Learn: Decision Trees*. URL: `https://scikit-learn.org/stable/modules/tree.html`.

[5] *Scikit-learn: k Nearest Neighbors*. URL: `https://scikit-learn.org/stable/modules/neighbors.html`.

[6] *Scikit-Learn: Preprocessing data*. URL: `https://scikit-learn.org/stable/modules/preprocessing.html`.

[7] *Scikit-learn: Random Forest Classifier*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`.