project2

April 16, 2020

1 Machine Learning in Python - Project 2

Due Wednesday, April 15th by 5 pm.

1.1 1. Setup

1.1.1 1.1 Libraries

[1]:
```
# Add any additional libraries or submodules below

# Display plots inline
%matplotlib inline

# Data libraries
import pandas as pd
import numpy as np

# Plotting libraries
import matplotlib.pyplot as plt
import seaborn as sns

# Plotting defaults
plt.rcParams['figure.figsize'] = (8,5)
plt.rcParams['figure.dpi'] = 80

from ipywidgets import interact


# sklearn modules
import pkg_resources
if pkg_resources.get_distribution("scikit-learn").version != '0.22.2':
    !pip install --upgrade scikit-learn
import sklearn

import sklearn.tree
import sklearn.ensemble
```

```
import sklearn.neighbors
import sklearn.preprocessing
from sklearn.metrics import mean_squared_error
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.linear_model import LogisticRegression

# import warnings
# warnings.filterwarnings('ignore')
```

```
Requirement already up-to-date: scikit-learn in /opt/conda/lib/python3.7/site-
packages (0.22.2.post1)
Requirement already satisfied, skipping upgrade: joblib>=0.11 in
/opt/conda/lib/python3.7/site-packages (from scikit-learn) (0.14.0)
Requirement already satisfied, skipping upgrade: scipy>=0.17.0 in
/opt/conda/lib/python3.7/site-packages (from scikit-learn) (1.3.1)
Requirement already satisfied, skipping upgrade: numpy>=1.11.0 in
/opt/conda/lib/python3.7/site-packages (from scikit-learn) (1.17.2)
```

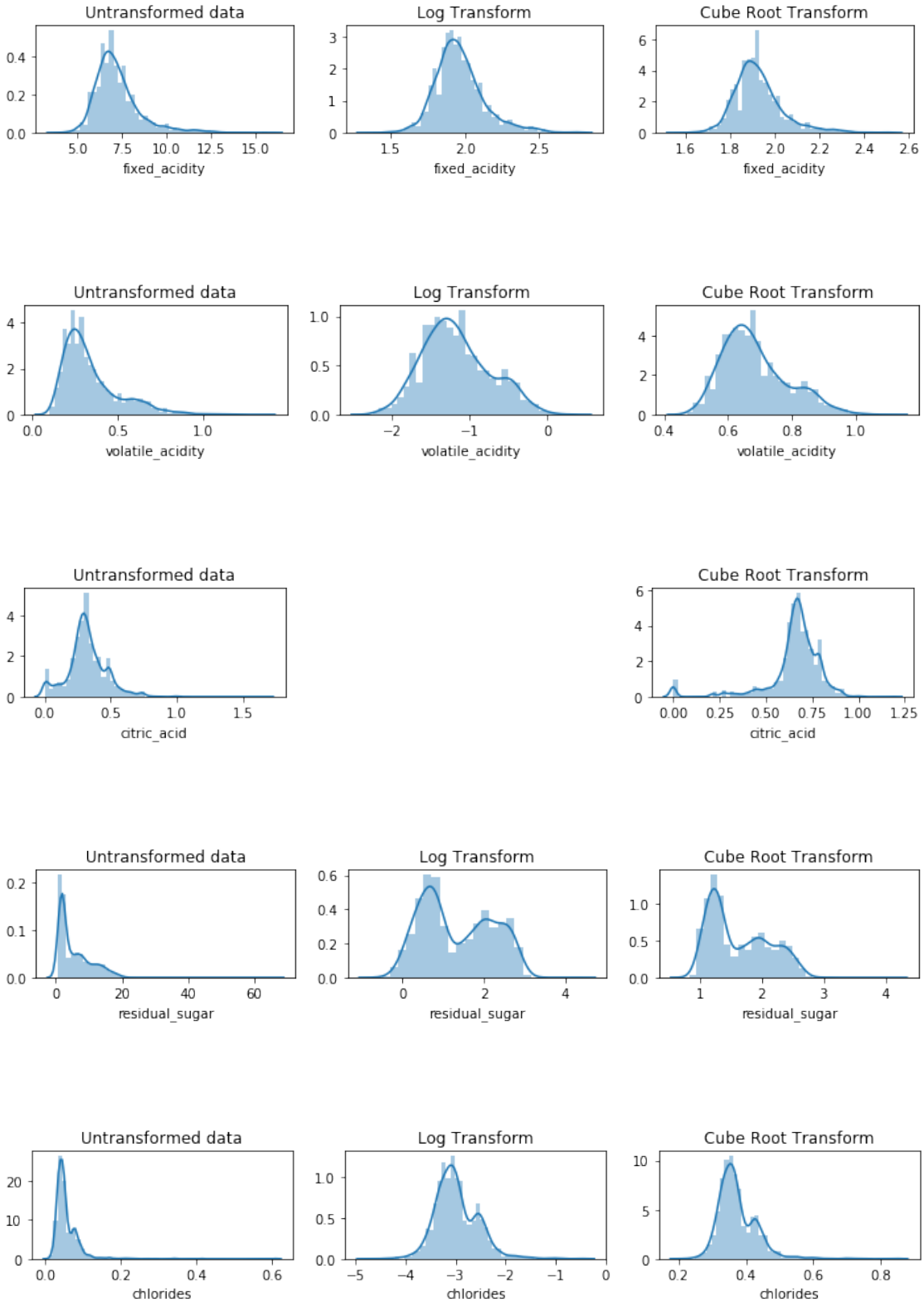
1.1.2 1.2 Data

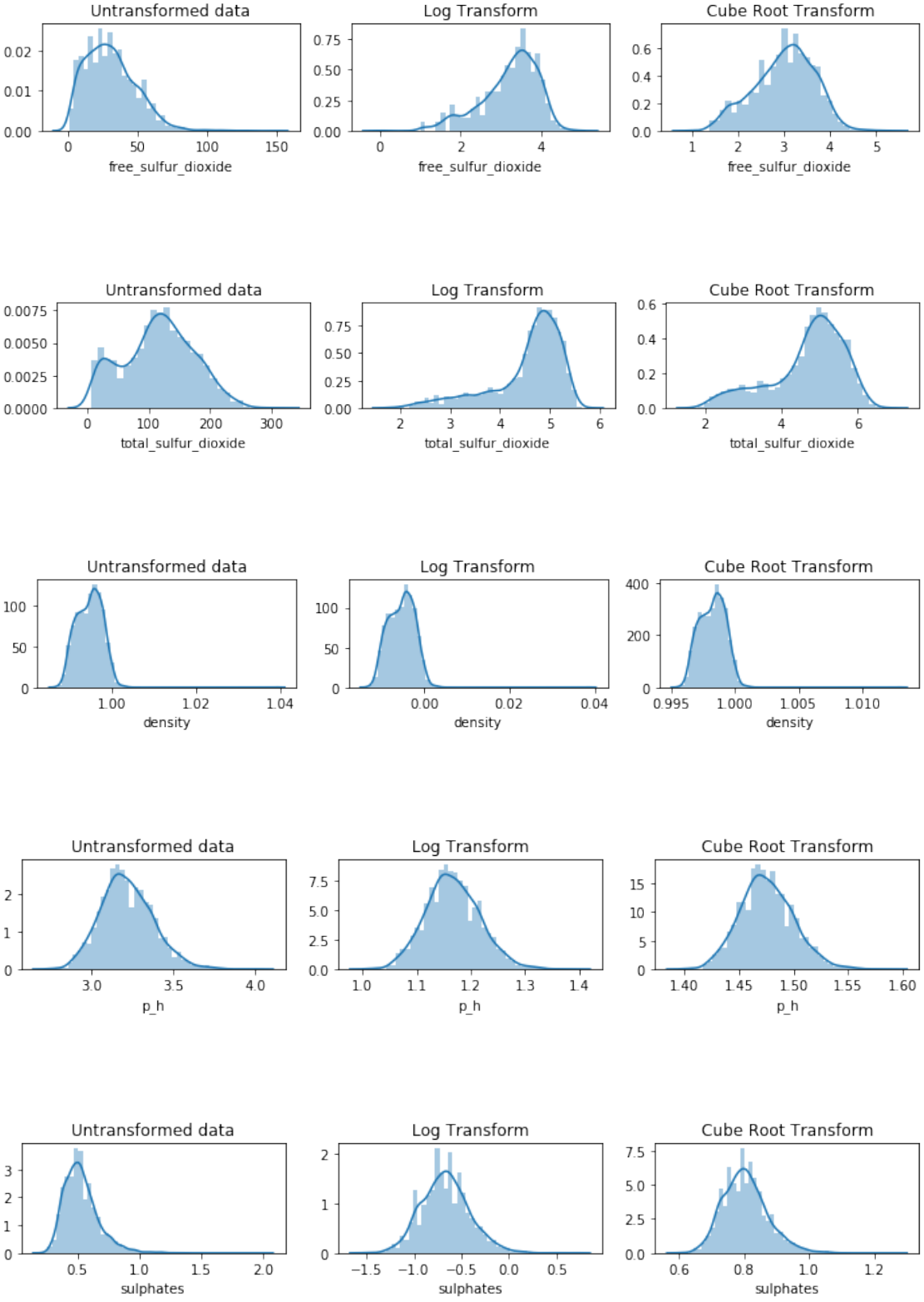
[2]:
```
wine_train = pd.read_csv("wine_qual_train.csv")
wine_test  = pd.read_csv("wine_qual_test.csv")
```

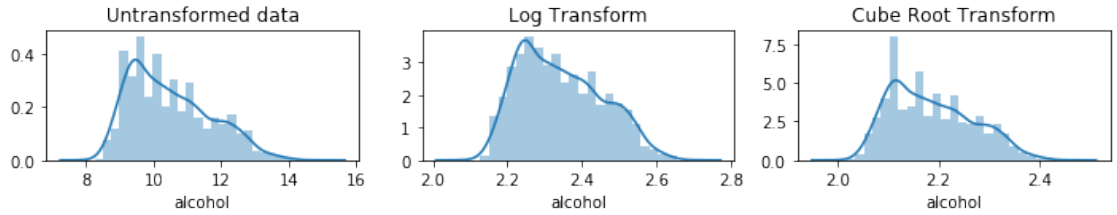
1.2 2. Exploratory Data Analysis and Preprocessing

[3]:
```
cats = ['fixed_acidity', 'volatile_acidity','citric_acid', 'residual_sugar',␣
 ↪'chlorides','free_sulfur_dioxide','total_sulfur_dioxide',␣
 ↪'density','p_h','sulphates','alcohol']
for i in cats:
    try:
        fig = plt.figure(figsize=(12,1.5))
        fig.tight_layout()
        plt.subplot(1,3,1)
        sns.distplot(wine_train[i])
        plt.title('Untransformed data')
        if i != 'citric_acid':
            plt.subplot(1,3,2)
            sns.distplot(np.log(wine_train[i]))
            plt.title('Log Transform')
        plt.subplot(1,3,3)
        sns.distplot((wine_train[i])**(1/3))
        plt.title('Cube Root Transform')
    except:
        print(i)
```

```
plt.show()
```


Untransformed data
Log Transform
Cube Root Transform
fixed_acidity
volatile_acidity
citric_acid
residual_sugar
chlorides


Untransformed data
Log Transform
Cube Root Transform
free_sulfur_dioxide
total_sulfur_dioxide
density
p_h
sulphates



[4]:
```
wine_train = pd.read_csv("wine_qual_train.csv")
wine_test  = pd.read_csv("wine_qual_test.csv")

def variable_transformer(df):
    for i in ['fixed_acidity', 'volatile_acidity','citric_acid',␣
 ↪'residual_sugar', 'chlorides','free_sulfur_dioxide','sulphates']:
        if i == 'free_sulfur_dioxide' or i == 'citric_acid':
            df[i] = df[i]**(1/3)
        else:
            df[i] = np.log(df[i])
    df['is_red'] = (df['wine'] == 'red').astype(int)
    df = df.drop(['wine'], axis = 1)
    return df

wine_train = variable_transformer(wine_train)
wine_test = variable_transformer(wine_test)

def quality_discretizer(df):
    names=[]
    for i in df['quality']:
        if i >= 7 and i<=10:
            names.append('excellent')
        elif i == 6:
            names.append('good')
        elif i == 5:
            names.append('average')
        elif i<= 4 and i>= 1:
            names.append('poor')
        else:
            names.append('Error')
    df['quality'] = names
    return df

wine_train = quality_discretizer(wine_train)
wine_test = quality_discretizer(wine_test)
```

In preprocessing, the first thing that was done was to convert the `quality` value from an integer into a rating, such as 'good' or 'average'. This was done using the function *quality_discretizer()*. Additionally, several other variable were transformed so that they would be more appropriate for fitting. Included is a plot showing the original distribution of every continuous variable in the data set. Many of these are 'badly' distributed, in that the distribution is skewed very differently from a Gaussian, which is normally distributed. Two transformations that can make distributions that are positively skewed are logarithms, and cube roots, so I have tried this for every variable and these results are also in the plot. It can be seen that `fixed_acidity`, `volatile_acidity`,`sulphates`, `residual_sugar` and `chlorides` are improved by the log transformation, `citric_acid` and `free_sulfur_dioxide` are improved by the cube root transformation, and `total_sulfur_dioxide`, `density`,`p_h`,`alcohol` are all either already decently good, or unhelped by either transformation (ie, `density`). A preliminary examination of the correlation matrix (stored in the supplementary materials), shows that we expect `volatile_acidity`, `chlorides`, `density` and `alchohol`.

1.3 3. Model Fitting and Tuning

[5]:
```
X = wine_train.drop(['quality'], axis=1)
y = wine_train.quality

Xt = wine_test.drop(['quality'], axis=1)
yt = wine_test.quality

parameters = {'n_estimators': range(1,82,10), 'max_depth': [None] + list(np.
 ↪arange(1,47,5))}
m_rf = GridSearchCV(sklearn.ensemble.RandomForestClassifier(), param_grid =␣
 ↪parameters, cv = 3)
m_rf.fit(X, y)
print(m_rf.best_params_)
```

```
{'max_depth': 11, 'n_estimators': 71}
```

The model that was chosen in the end was a Random Forest model. This was chosen over Logistic Regression, SVM, Decision Trees, K Nearest Neighbours, and Radius Nearest Neighbours, as seen in the supplementary materials (see the table). None of these methods obtained an accuracy of more than 58% (SVM), so this is the main reason that the Random Forest model is used, which achieved an accuracy of 64% on the test data.

Model	Accuracy
SVM	58%
Logistic Regression	55%
Decision Trees	56%
Random Forest	65%
KNN	50%
RNN	45%

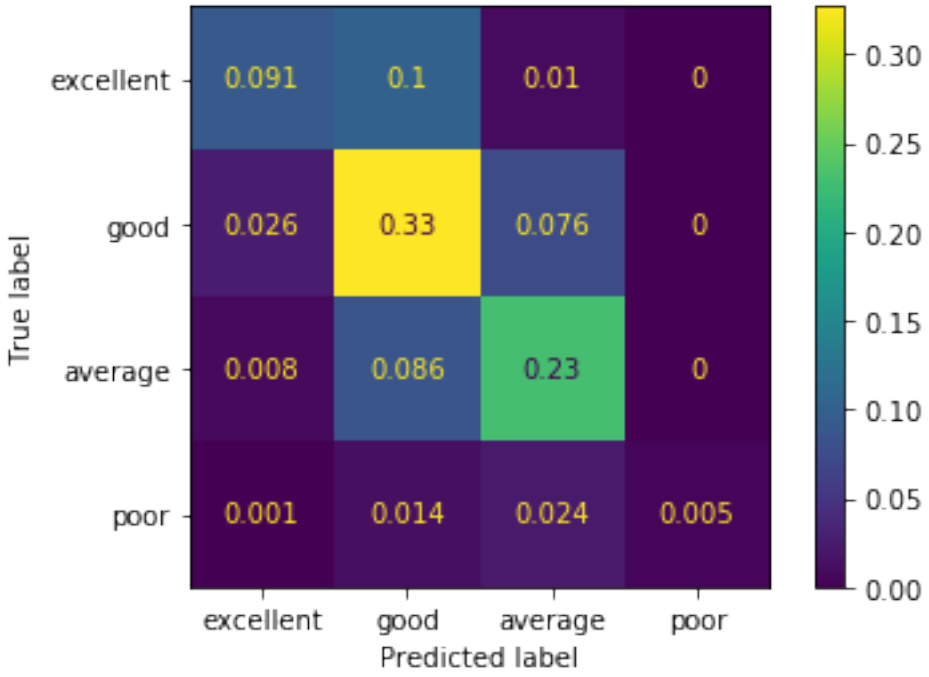
Once this model was chosen, its parameters, `n_iterations` and `max_depth`, were optimised using a gridsearch. This was done to avoid the potential overfitting that would occur if this method were done by hand - for example it is possible to obtain 65% accuracy for certain values of the parameters, but this is not a consistent result and applies only to this specific data set. This consistently selected as high an `n_iterations` value as allowed, but any improvements in the score of the model are negligible once a value of 30-40 is reached. Therefore the value `n_iterations = 80` was used going forward. Again, for `max_depth`, the results are not always consistent, although higher values were consistently picked, and the most common result is `max_depth = None`, so this will be used too. Due to the discrete nature of the data trained on and the relatively small sample set, the results of the gridsearch to depend highly on the cross validation folds used, leading to inconsistent results. This is why some human judgement was used to aggregate this. From a modelling prespective, it makes sense to have a large amount of estimators as this will increase the consistency of the model (as there are more chances to get the right answer). This is also not a simple input space, with 12 variables to account for, so a large tree depth also makes sense.

1.4 4. Discussion

[6]:
```
n_estimators=80
max_depth=None
normalize='all'
qualities = ['excellent','good','average','poor']
X = wine_train.drop(['quality'], axis=1)
y = wine_train.quality
m_rf = sklearn.ensemble.RandomForestClassifier(
    n_estimators=n_estimators, max_depth=max_depth
).fit(X,y)

res = wine_test.copy()
Xt = wine_test.drop(['quality'], axis=1)
res["pred_label"] = m_rf.predict(Xt)


sklearn.metrics.plot_confusion_matrix(m_rf, Xt, res.quality,␣
 ↪include_values=True, normalize=normalize, labels=qualities)
plt.show()
print(
    sklearn.metrics.classification_report(res.quality, res.pred_label)
)
```



	precision	recall	f1-score	support
average	0.68	0.71	0.69	325
excellent	0.72	0.45	0.55	202
good	0.62	0.76	0.68	429
poor	1.00	0.11	0.20	44
accuracy			0.65	1000
macro avg	0.75	0.51	0.53	1000
weighted avg	0.68	0.65	0.64	1000

Overall, this model has an accuracy of 65%. This means that there is a 65% chance that a given data point will be classified correctly. While this is not amazing, it is definitely helpful, and there is more to it than there first seems. Importantly, the type of misclassification it makes is almost always a relatively 'harmless' mistake, in that the estimate of the quality is only off by one. This can be seen in the confusion matrix plot, where almost all of the errors are due to misclassifying 'excellent' or 'average' data as 'good', or misclassifying 'good' data as 'average'. The error rates for more serious misclassifications are much lower, only 3.2%. This is important, because while not ideal, misclassifying an 'excellent' wine as 'good' will lead to some small revenue loss, misclassifying a 'poor' wine as 'excellent' would seriously impact the trust that people have in the company that use this model. This means the accuracy of the model to make a close enough classification so that it is only one out it in fact over 96%, which is very good. As can be seen from the ROC curve in the supplementary materials, the model seems to classify 'average' quality wines well, but has problems with other types. (It is also important to note that due to the Randomness of the Random Forest

Model, the exact numbers given here may not be accurate when this is rerun, but they should be roughly representative of the results obtained regardless.)

Below, the model is retrained, with preestablished parameters, on the full data set given.

[7]:
```
wine_comb = pd.concat([wine_train,wine_test], axis=0)
X_comb = wine_comb.drop(['quality'], axis=1)
y_comb = wine_comb.quality
m_rf_final = sklearn.ensemble.RandomForestClassifier(
    n_estimators=80, max_depth=None
).fit(X_comb,y_comb)
```

1.5 5. Model Validation

[8]:
```
wine_holdout = pd.read_csv("wine_qual_holdout.csv")

wine_holdout = variable_transformer(wine_holdout)
wine_holdout = quality_discretizer(wine_holdout)

X_holdout = wine_holdout.drop('quality', axis=1)
y_holdout = wine_holdout.quality
```

[9]:
```
sklearn.metrics.confusion_matrix(y_holdout, m_rf_final.predict(X_holdout))
```

[9]:
```
array([[325,   0,   0,   0],
       [  0, 202,   0,   0],
       [  0,   0, 429,   0],
       [  0,   0,   0,  44]])
```

[10]:
```
print(
    sklearn.metrics.classification_report(y_holdout, m_rf_final.
 ↪predict(X_holdout))
)
```

```
              precision    recall  f1-score   support

     average       1.00      1.00      1.00       325
   excellent       1.00      1.00      1.00       202
        good       1.00      1.00      1.00       429
        poor       1.00      1.00      1.00        44

    accuracy                           1.00      1000
   macro avg       1.00      1.00      1.00      1000
weighted avg       1.00      1.00      1.00      1000
```

As can be seen here, despite the efforts gone to, the accuracy for the models has not been improved above 63-5%. This means that, although the model may be of some use in providing early estimates

of a wine's quality - due to its accuracy being no more than one quality classification off in the vast majority of cases - this model cannot serve as a replacement for human input in the decision of classification. It may also be true that due to the subjective nature of human taste, this adds a degree of difficulty to this modelling problem that is not always present in other circumstances.