Solution 1 - Part 2 (b)

Wolfram Wiesemann

MSc Business Analytics – Machine Learning

1 Importing the Dataset

If you are using Jupyter Notebook, you have to navigate to the folder in which you have stored the data first:

```
%cd myfolder
%ls
```

These are some 'magic' commands for the Jupyter environment that are not Python commands. Search online for 'jupyter magic commands' to learn more about them!

We will use Pandas to import the data set. By manually exploring the data file first, we can see that columns are separated by a semicolon.

```
import pandas as pd
df = pd.read_csv ('winequality-red.csv', sep = ';')
```

This is a clean data set without any missing values etc. You can explore the data set via:

```
df.ix[5]
df.ix[5].quality
df.head (10)
df[2:5]
df.columns
df.describe()
df.corr()
```

2 Creating a New Binary Column for Good Wines

We aim to classify wines in a binary fashion. The "quality" column is a number between 1-10. We create the new binary column as follows:

```
def goodwine (quality):
    if quality >= 6:
        return 1
    return 0
df['GoodWine'] = df.quality.apply (goodwine)
```

Alternatively, you can do it in one line by using the lambda syntax:

```
df['GoodWine'] = df.quality.apply (lambda x : 1 if x >= 6 else 0)
```

3 Moving the Data to NumPy Arrays

SciKit-Learn takes its input in the form of NumPy arrays. Our features are the first 11 columns, and our classification label is the 13th column:

```
X = np.array (df[df.columns[:11]])
y = np.array (df.GoodWine)
print X.shape
print y.shape
```

4 Splitting the Data Set

We have 1,599 data points. We are not going to create a validation set as we are using cross-validation to select the best k for our classifier.

Tip: Data Shuffling

Sometimes datasets are ordered in a non-random way. You should always make sure that your training set, validation set and test set come from the same distribution. Thus, the data should be shuffled before splitting.

```
from sklearn.utils import shuffle
X,y = shuffle(X, y)
```

Now we can split the data set manually:

```
X_train_unproc = X[:959] # before preprocessing
X_test_unproc = X[959:]
y_train = y[:959]
y_test = y[959:]
```

Alternatively, we can use the pre-built '`train_test_split`' function from sklearn:

```
from sklearn.cross_validation import train_test_split
X_train_unproc, X_test_unproc, y_train, y_test = train_test_split (X, y,
    test_size=0.4)
```

which does the shuffling and splitting all at once.

5 Data Normalisation

We are going to normalise the data as the Nearest Neighbours classifier is sensitive to scaling. While here the training set and the test set are available in advance, we are going to take a general approach: We normalise according to the training data and apply the same transformation to the test set in a consistent manner.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit (X_train_unproc)
X_train = scaler.transform (X_train_unproc)
X_test = scaler.transform (X_test_unproc)
```

The last command can be run after training, should the test data be initially unavailable.

6 Loading and Training the Classifier

First we need to import the Nearest Neighbours package:

```
from sklearn.neighbors import KNeighborsClassifier
```

To test the classifier, we first train it with $k = 3$:

```
clf = KNeighborsClassifier (3) # initialize
clf.fit (X_train, y_train) # train using training set
```

Tip: Lazy Learners

Nearest Neighbours is a 'lazy' classifier. At training stage, the classifier only stores the data as well as the parameter k.

To evaluate the performance of your classifier on the training data, we type:

```
clf.score (X_train, y_train)
```

We obtain a performance of 0.857. Note that due to the random shuffling you might obtain a different number. For our initial choice of $k = 3$, we now estimate the performance on new data using cross-validation.

Tip: Avoid Data Snooping

Unless you have made up your mind that $k = 3$ is the final classifier that you are going to use (for example for your coursework), you should *not* use the test set for performance evaluation. If you have split the data into a training set and a test set, as we have done, then you will need to use cross-validation to select the best k using the training data. Only use the test set for the final estimate!

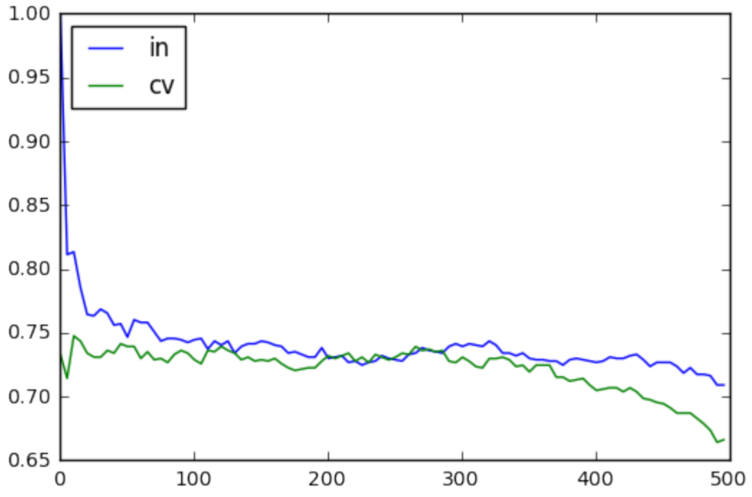
```
from sklearn import cross_validation
scores = cross_validation.cross_val_score (clf, X_train, y_train, cv = 5)
score = scores.mean()
```

The 'scores' object now contains an array of 5 numbers. You can take the average to obtain an estimate of your classifier's performance on future data. We obtain a value of 0.715.

7 Selecting the Best k

We are going to use cross-validation to choose the value of k that has the most promising performance on future data:

```
ks = range (1, 501, 5)  # try k 1,6,11,...501
inSampleScores = []
crossValidationScores = []
for k in ks:
    clf = KNeighborsClassifier(k).fit (X_train, y_train)
    inSampleScores.append (clf.score (X_train, y_train))
    scores = cross_validation.cross_val_score (clf, X_train, y_train, cv = 5)
    crossValidationScores.append (scores.mean())
p1 = plt.plot (ks, inSampleScores)
p2 = plt.plot (ks, crossValidationScores)
legend(['in', 'cv'], loc = 'upper left')
```



As we have seen in class, for small (large) values of k the Nearest Neighbours classifier suffers from overfitting (underfitting). Our plot indicates that $k = 80$ is a reasonable choice.

8 Evaluating the Selected Classifier on the Test Set

We can now evaluate the classifier with $k = 80$ on the test set:

```
clf = KNeighborsClassifier(80).fit (X_train, y_train)
y_test_pred = clf.predict (X_test)
```

As we also have the actual outputs `y_test`, we can calculate the performance:

```
clf.score (X_test, y_test)
```

or manually via

```
succ = np.sum (np.where (y_test == y_test_pred, 1, 0))
total = y_test.shape[0]
score = float (succ) / total
print score
```

We obtain a value of 0.36. Apart from just a score, we can generate further statistics via:

```
from sklearn.metrics import classification_report
print classification_report (y_test , y_test_pred , target_names = iris.
   target_names)
```

and

```
from sklearn.metrics import confusion_matrix
print confusion_matrix (y_test , y_test_pred)
```

About half of the wines in our test set are good. Our Nearest Neighbours classifier improved the ‘random guessing’ performance of 50% to 73%.