Project - k nearest neighbours on the TunedIT data set

Data Programming with Python

We will study the method of k nearest neighbours applied to a music classification data set. These data come from the TunedIT website `http://tunedit.org/challenge/music-retrieval/genres`. Each row corresponds to a different sample of music from a certain genre. The original challenge was to classify the different genres (the original prize for this was hard cash!). However we will just focus on a sample of the data (∼4000 samples) which is either rock or not. There are 191 characteristics (go back to the website if you want to read about these) The general tasks are as follows:

- Load the data set
- Standardise all the columns
- Divide the data set up into a training and test set
- Write a function which runs k nearest neighbours (kNN) on the data set.
  (Don't worry you don't need to know anything about kNN)
- Check which value of k produces the smallest misclassification rate on the training set
- Predict on the test set and see how it does

1. Load in the data using the pandas `read_csv` function. The last variable *RockOrNot* determines whether the music genre for that sample is rock or not. What percentage of the songs in this data set are rock songs? (1 indicates the song is a rock song, 0 indicates that it is not)

2. To perform a classification algorithm, you need to define a classification variable and separate it from the other variables. We will use *RockOrNot* as our classification variable. Write a piece of code to separate the data into a `DataFrame` X and a `Series` y, where X contains a standardised version of everything except for the classification variable (*RockOrNot*), and y contains only the classification variable. To standardise the variables in X, you need to subtract the mean and divide by the standard deviation.

3. Which variable in X has the largest correlation with y?

4. When performing a classification problem, you fit the model to a portion of your data, and use the remaining data to determine how good the model fit was. Write a piece of code to divide X and y into training and test sets, use 75% of the data for training and keep 25% for testing. The data should be randomly selected, hence, you cannot simply take the first, say, 3000 rows. Use the seed 123 when generating random numbers.
   Note: The data may not spilt equally into 75% and 25% portions. In this situation you should round to the nearest integer.

5. What is the percentage of rock songs in the training dataset and in the test dataset? Are they the same as the value found in question 1?

6. Now we're going to write a function to run kNN on the data sets. kNN works by the following algorithm:

   (a) Choose a value of k (usually odd)
   (b) For each observation, find its k closest neighbours
   (c) Take the majority vote (mean) of these neighbours
   (d) Classify observation based on majority vote

   We're going to use standard Euclidean distance to find the distance between observations, defined as $\sqrt{(x_i - x_j)^T(x_i - x_j)}$. A useful short cut for this is the scipy functions `pdist` and `squareform`.

   The function inputs are:

   - `DataFrame` X of explanatory variables
   - binary `Series` y of classification values
   - value of k (you can assume this is always an odd number)

   The function should produce:

   - `Series` y_{star} of binary predicted classification values

   (A sketch of the function is given in the *.py* template.)

7. The misclassification rate is the percentage of times the output of a classifier does not match the classification value. Calculate the misclassification rate of the kNN classifier for X_{train} and y_{train}, with $k = 3$.

8. The best choice for k depends on the data. Write a function `kNN_select` that will run a kNN classification for a range of k values, and compute the misclassification rate for each.

   The function inputs are:

   - `DataFrame` X of explanatory variables
   - binary `Series` y of classification values
   - a list of k values k_{vals}

   The function should produce:

   - a `Series` *mis_class_rates*, indexed by k, with the misclassification rates for each k value in k_{vals}.

9. Run the function `kNN_select` on the training data for $k = [1, 3, 5, 7, 9]$ and find the value of k with the best misclassification rate. Use the best value of k to report the misclassification rate for the test data. What is the misclassification percentage with this k on the test set?

10. Write a function to generalise the kNN classification algorithm. The function should:

    - Separate out the classification variable for the other variables in the dataset, i.e. create X and y.
    - Divide X and y into training and test set, where the number in each is specified by *percent_train*.

- Run the k nearest neighbours classification on the training data, for a set of k values, computing the misclassification rate for each k
- Find the k that gives the lowest misclassification rate for the training data, and hence, the classification with the best fit to the data.
- Use the best k value to run the k nearest neighbours classification on the test data, and calculate the misclassification rate

The function should return the misclassification rate for a k nearest neighbours classification on the test data, using the best k value for the training data You can call the functions from question 6 and 8 inside this function, provided they generalise, i.e. will work for any dataset, not just the TunedIT data set.

Test your function with the TunedIT data set, with `class_column` = '*RockOrNot*', seed = the value from Q4, *percent_train.* = 0.75, and k_{vals} = set of k values from Q8, and confirm that it gives the same answer as Q9.

Now test your function with another dataset, to ensure that your code generalises. You can use the `house_votes.csv` dataset, with *Party* as the classifier. Select the other parameters as you wish. This dataset contains the voting records of 435 congressman and women in the US House of Representatives. The parties are specified as 1 for democrat and 0 for republican, and the votes are labelled as 1 for yes, -1 for no and 0 for abstained. Your kNN classifier should return a misclassification for the test data (with the best fit k value) of 8%.

All of your code and text answers should be written into the *.py* template. Save your filled *.py* file with the following name structure *SurnameFirstname_Project.py* (where *Surname* and *Firstname* should be replaced with your name) and upload it to Brightspace. Additionally, you must upload a PDF of your code. Create a PDF from Canopy by selecting *File* $\rightarrow$ *Print*, and print to PDF. Unsure that all of your code and text answers are included in both your *.py* file and the PDF.