Data preprocessing with scikit-learn

One of the keys that determine how well a machine learning algorithm can learn is the quality of data and the amount of useful information it contains. In this project we will explore various data preprocessing techniques that will be useful in building good machine learning models.

You have been provided with a dataset that contains different characteristics of various cars and their associated insurance risk rating. This dataset has a total of 26 features, although some instances of these features have missing values. You will find the labels for these features in the file '*feature_labels.txt*' and the corresponding feature values in the file '*dataset.txt*'. The missing values are denoted by the '?' symbol. You have also been provided with a python script '*main_script.py*', which is where you will write all your code.

For sections (a) to (g) you should save your results in the variables specified in *main_script.py* so that we can autograde your work.

(a) Your first task is to generate a data frame from the dataset we provided you using the pandas module. Your data frame should be similar to the one shown in Figure 3.1 below, where missing values are represented by NaN.

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...
3	2	164.0	audi	gas	std	four	sedan	fwd	front	99.8	...
4	2	164.0	audi	gas	std	four	sedan	4wd	front	99.4	...
5	2	NaN	audi	gas	std	two	sedan	fwd	front	99.8	...
6	1	158.0	audi	gas	std	four	sedan	fwd	front	105.8	...
7	1	NaN	audi	gas	std	four	wagon	fwd	front	105.8	...
8	1	158.0	audi	gas	turbo	four	sedan	fwd	front	105.8	...
9	0	NaN	audi	gas	turbo	two	hatchback	4wd	front	99.5	...
10	2	192.0	bmw	gas	std	two	sedan	rwd	front	101.2	...

Figure 3.1 Data Frame illustrating some instances from our dataset

You should return the number of missing values for every feature **as a python list**. For instance, in Figure 3.1 symboling has zero missing values, normalized-losses has six missing values and so forth, therefore the list returned should be [0, 6, 0, 0, ...]. Save your list in the variable "*missing_features*". **– 10 points**

(b) Your next task is to fill in the values for the missing features, but in order to do that you will first find the averages for every feature that has missing values. The averaging methods we will use for this section are the mode, the median and the mean.

You will find the mode for the following features:

num-of-doors

You will find the mean for the following features:

normalized-losses, price, bore, stroke

You will find the median for the following features:

horsepower, peak-rpm

After finding these different averages you should now modify your data frame by replacing the missing values with the corresponding averages you calculated above. Save your data frame after making these changes in the variable df_b. – **15 points**

(c) Map the values of the ordinal features *num-of-doors* and *num-of-cylinders* with their integer representation. For instance, if the value of the feature is 'eight', replace this word representation with the corresponding integer 8. Save your data frame after making these changes in the variable df_c. – **10 points**

(d) Identify the nominal features in your dataset and perform one-hot encoding on these features. For example, the feature fuel-type will be transformed to the features fuel-type_gas and fuel-type_diesel. Modify your data frame and replace its nominal features with their one-hot encoding representation and save it to the variable df_d. – **10 points**

Note

Be careful when saving your data frames to the variables df_b, df_c, and df_d. Make sure these variables are not referencing the same data frame object. For instance the variable df_b should only reflect the changes in your data frame after the tasks in section (b) have been completed, and it should not be modified after the tasks in sections (c) and (d) have been completed. The variable df_c should reflect the changes after the tasks in sections (b) and (c) have been completed and finally the variable df_d should reflect the changes after the tasks in sections (b), (c) and (d) have been completed.

(e) The label '*symboling*' from our dataset represents the insurance risk rating for every automobile where +3 indicates that the car is very risky and -3 indicates that it is pretty safe. We will use this label as the class label we wish to predict. Using the method *train_test_split* from *scikit-learn*, partition the dataset into a training and a test set, with the parameters *random_state* = 0 and *test_size* = 0.3. After that, train a decision tree classifier with maximum depth of 20, random_state of 0 and 'entropy' as the measure for information gain. Find the accuracy on the training and the test set and save them to the varibles training_accuracy and test_accuracy. – **10 points**

(f) For this section you will build a random forest with the goal of assessing feature importance. Train a random forest classifier with the parameters *n_estimators* = 10, *random_state* = 0 and *max_depth*=20. Use this forest to assess how much each feature contributes to the model, and return a list of tuples where each tuple comprises of a feature label and its contribution score. Save this list of tuples in the variable feature_importance – **10 points**

(g) You will now train multiple decision trees using various number of features and the same parameters as in part (e). You will first train a decision tree with the most important feature and record its (test) accuracy, then train it with the first two important features and record that accuracy, then the first three important features, until you have exhausted all the features. You will then plot a line graph with accuracy on the vertical axis and the number of features on the horizontal axis (if we call the function *visualise_line_graph* it should generate this plot). Finally, find the number of features that produce the highest accuracy. Save this result in the variable num_of_features. – **15 points**

(h) For this section you will write a function called *custom_model_prediction* (see *main_script.py*). This function will take a text file as an input, and this text file represents a test set. This text file will have the same format as the original *dataset.txt* file except that the symboling (the class label) column will be removed. It may also have some missing values for some features (in this case *num-of-doors, normalized-losses, price, bore, stroke, horsepower, peak-rpm*), so you must also handle those in any way you see fit. The file *sample_test.txt* shows the format of this file and contains only 2 instances, but we will test your program with any number of instances so you should take that into account. You are free to use any classifier and any feature extraction methods of your choice. This function should just return predictions (as a list) of every instance in the test set. – **20 points**