Data Mining
Assignment 3

Ali Gholami

Department of Computer Engineering & Information Technology
Amirkabir University of Technology

https://aligholamee.github.io
aligholami7596@gmail.com

Abstract

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, its also widely used in machine learning, which will be the main focus of this article.

Keywords. *Decision Tree, Normalization, Generalization, Preprocessing, Feature Engineering, Scikit-Learn, Pandas, Numpy, Python 3.5.*

1 Data Preprocessing

In this section, we'll be looking at our training data from different aspects. First, we need to get a quick intuition of how data looks like, how is that distributed and what to do with that! To do this, we'll be using some functions as described below.

```
separate_output('Training Data Types')
print(train_data.dtypes)
```

In this part, we have printed the data types of our training set. Note that *separate-output* is a self-defined function to make thing more clear in the terminal. Now, its time for some statistics. To get a full understanding of how our numerical data is distributed, we use the following code.

```
separate_output('Statistical Information')
print(train_data.describe())
```

The result of this part of code will be some statistical parameters such as: *variance, mean, max, min, counts*. These can be useful in the future to make decisions about *data normalization*. Another amazing feature that *Pandas* has provided for us is the ability to separately describe each column in the dataset. As an example, the first column contains 686 missing values. Use the following code to believe this fact.

```
separate_output('Counts Values on a Column')
print(train_data['col_1'].value_counts())
```

This column may not be seem much useful at the first glance, but we keep it since there are some good values for that column which might make it useful while we go further in the classification task. Some of these columns are completely useless. Let's find them. The following function will return a dictionary consisting of number of missing values of each column.

```
def compute_nans(df):

nans_dict = {}

for col in df:
        nan_col_counter = 0
        for row in df[col]:
                if(row == '?'):
                        nan_col_counter += 1
        nans_dict[str(col)] = [nan_col_counter]

return nans_dict
```

We can obviously remove the following columns with more than 500 missing values.

```
cols_to_drop = [key for key, value in nan_cols.items() if value > 500]
train_data = train_data.drop(cols_to_drop, axis=1)
```

References

[1] Prashant Gupta, *Cross-Validation in Machine Learning*. Towards Data Science, Jun 5, 2017.

[2] Scikit-Learn, *sklearn.tree.DecisionTreeClassifier*. http://scikit-learn.org.

[3] Scikit-Learn, *Tuning the hyper-parameters of an estimator*. http://scikit-learn.org.