INF582: Data Science – Learning from Data
École Polytechnique
Lab 2: Feature Selection

Jean-Baptiste Bordes, Fragkiskos Malliaros, Nikolaos Tziortziotis and Michalis Vazirgiannis

December 11, 2015

1 Description

The goal of this lab is to study *feature selections* techniques and to examine how they can be applied on real data. Initially, we briefly discuss the importance of feature selection in data analysis, and then we describe the tasks that need to be performed in the lab.

2 Introduction to Feature Selection

In machine learning and data analysis, *feature selection* is the process of selecting a subset of relevant features for use in the construction of a learning model. The central assumption when using a feature selection technique is that the data contains many redundant or irrelevant features. Redundant features are those which provide no more information than the currently selected features, and irrelevant features provide no useful information in any context. That way, feature selection helps a learning model (e.g., classification) by choosing features that will give as good or better accuracy whilst requiring less data.

Note that, feature selection is different from the task of dimensionality reduction which described in the previous lab. Both methods seek to reduce the number of attributes in the dataset; however, a dimensionality reduction method aims to create a representation of the data in a new space preserving some of the underlying properties of the data (e.g., variance), whereas feature selection methods include and exclude attributes present in the data without changing them.

Feature selection techniques can provide several benefits when constructing learning models. First of all, these methods help the model to avoid overfitting, leading to enhanced generalization. By selecting the most important and relevant features from a dataset, the accuracy of a learning model can also be enhanced, since a lot of the noisy features are excluded from the process. Moreover, feature selection methods can reduce the training time of a model, that in many cases it can be a computational bottleneck in the learning process. In general, less attributes is desirable because it reduces the complexity of the model, and a simpler model is simpler to understand and explain (improved model interpretability).

A feature selection algorithm can be seen as the combination of a search technique for proposing new feature subsets, along with an evaluation measure which scores the different feature subsets. The simplest algorithm is to test each possible subset of features finding the one which minimizes the error rate. This is an exhaustive search of the space, and it can be computationally intractable. There are two main categories of feature selection techniques: **wrapper methods** and **filter methods**:

- *Wrapper methods* use a search algorithm to search through the space of possible features and evaluate each subset by running a model on the subset. In other words, each new subset is used to train a model, which is tested on a test set. Counting the number of errors made on that test set (the error rate of the model) gives the score for that subset. Wrappers can be computationally expensive and have a risk of over fitting to the model.

- *Filter methods* use the same search strategy as wrappers, but instead of evaluating against a model, a simpler filter measure is evaluated. This measure is chosen to be fast to compute, while still being able to capture important information of the data. Examples of such measures include the *mutual information*, the χ^2 *test* and the *Pearson correlation coefficient*.

Filters are usually less computationally intensive than wrappers, but they produce a feature set which is not tuned to a specific type of predictive model. This lack of tuning means a feature set produced by a filter method is more general than the set from a wrapper, usually giving lower prediction performance than a wrapper. However, the feature set does not contain the assumptions of a prediction model, and therefore is more useful for exposing the relationships between the features. Additionally, typically filter methods provide a feature ranking rather than an explicit best feature subset.

Next, we describe the measures of χ^2 and information gain that will be used in the lab.

2.1 χ^2 Measure

A very well known measure for feature selection is the χ^2 *measure*. The χ^2 test is widely used in statistics to test the independence of two events. That is, two events A and B are considered to be independent if $P(AB) = P(A)P(B)$, or equivalently if $P(A|B) = P(A)$ and $P(B|A) = P(B)$. In feature selection, the events correspond to the occurrence of an attribute (feature) and to the occurrence of a class and χ^2 is used to test whether these two are independent. Then, the features are ranked according to the following formula:

$$\chi^2(A) = \sum_{i=1}^{|N|} \sum_{j=1}^{|C|} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}, \tag{1}$$

where $|N|$ is the number of instances, $|C|$ the number of classes, O_{ij} is the observed frequency of class j for feature A and E_{ij} is the expected frequency which is given from the following formula:

$$E_{ij} = \frac{(\text{\# of instances with value i}) \cdot (\text{\# of instances in class j})}{|N|}.$$

2.2 Information Gain

In Information Theory, *entropy* is the average amount of information contained in each message received. In the case of feature selection, let Y be the random variable that corresponds to the class label. Then, the entropy of class Y characterizes the purity of the class and is given by the following formula:

$$\text{Entropy}(Y) = -\sum_{c=1}^{v} (P(y_c) * \log_2 P(y_c)),$$

where v all the possible class labels and $P(y_c)$ the probability of the class label c. For example, suppose that we have to deal with a dataset of 16 instances, and each of them belongs to one of the two possible classes: $\{Y, N\}$ (Yes, No). Let $R = \{Y, N, Y, N, Y, Y, Y, Y, N, N, Y, N, N, N, Y, Y\}$ be the set of the class labels. Then, the entropy will be equal to: $\text{Entropy}(R) = \left[\left(\frac{9}{16}\right) \log_2 \left(\frac{9}{16}\right) + \left(\frac{7}{16}\right) \log_2 \left(\frac{7}{16}\right)\right] = 0.9836$.

In this case, entropy is very close to one, which is expected since each class contains almost half part of the dataset.

The measure of *information gain* is used to compute the importance of a feature with respect to the class labels. More precisely, the information gain (IG) is defined as the expected reduction in entropy caused by partitioning the instances of the data according to a given feature. Let Y be the class labels and A be the feature of interest. Then, IG(Y, A) is given by the following formula:

$$\text{IG}(Y, A) = \text{Entropy}(Y) - \sum_{u \in Values(A)} \frac{|A_u|}{|A|} Entropy(A_u), \tag{2}$$

where $A_u, \forall u \in Values(A)$ are the sets after the splitting of the dataset according to the possible values of attribute A.

3 Evaluation of Feature Selection Methods

The goal of this lab is to study the effects of feature selection techniques in learning tasks. More precisely, we will apply the χ^2 and Information Gain measures to examine their performance in a classification task.

3.1 Description of the dataset

The dataset (`data.csv`) describes a set of 102 molecules of which 39 are judged by human experts to be musks (class 1) and the remaining 63 molecules are judged to be non-musks (class 2). The goal is to learn to predict whether new molecules will be musks or non-musks. However, the 166 features that describe these molecules depend upon the exact shape, or conformation, of the molecule. Because bonds can rotate, a single molecule can adopt many different shapes. To generate this data set, all the conformations of the molecules were generated to produce 6598 conformations. Then, a feature vector was extracted that describes each conformation. That way, the final dataset has 6598 instances and 166 features.

3.2 Pipeline of the task

The goal is to train a classifier to predict the label of a new molecule (musk or non-musk). In our case, we will use a *logistic regression*[1] classifier. We will apply feature selection methods to select a subset of the 166 features, and then the predictive power of the new feature set will be evaluated on the classifier. Next, we describe the pipeline of the lab which is included in the `main.py` Python script.

In the first part of the pipeline, we load the data and we extract the class labels. That way, variable X will contain the actual data and variable Y the class labels.

```
# Load the data set
data = loadtxt('data.csv', delimiter=',')
#load 1st column
Y = data[:,0:1]
# load columns 2 - end
X = data[:,1:data.shape[1]]
```

Then, the basic part of the pipeline follows. As this point, we enable or disable feature selection setting `True` or `False` the `featureSelection` variable. The idea is to examine the performance of

[1] Wikipedia's lemma for *logistic regression*: `http://en.wikipedia.org/wiki/Logistic_regression`.

the classifier for both cases – with and without feature selection. For the first case, we also need to set the number of features that will be selected (`num_feat` variable). For feature selection, we apply the χ^2 and Information Gain measures, as provided by the `chiSQ(X,Y)` and `infogain(X,Y)` functions; both of them have been implemented in the files `chiSQ.py` and `infogain.py`. Then, the features are ranked based on their importance (as returned by the feature selection techniques).

```
# Enables or disables feature selection
featureSelection = True

if featureSelection == True:
# number of features to be considered in the classification
# should be changed to compare performance for different number
# of features
num_feat = 50

# for each feature we get its feature selection value (x^2 or IG)
# gain = infogain(X,Y)
gain = chiSQ(X,Y)

index = argsort(gain)[::-1]

# select the top num_feat features
X = X[:,index[:num_feat]]

X = transpose(X)
```

The function `logisticRegression(X,Y)` trains the Logistic Regression classifier using the data that are stored on matrix X and the labels Y. In general, Logistic Regression is a type of regression that predicts the probability of ocurrence of an event by fitting data to a logit function (logistic function)[2]. In this lab, we consider the classification model as a black box that returns the class label of a new molecule. The parameters of the classifier after the training process are stored in variable `w`.

```
w = logisticRegression(X,Y)
```

Since the classifier has been trained, it can be used to predict the class of new data instances (i.e., molecules in our case). The data that will be used for prediction are stored in the `test.csv` file.

```
# test data
data = loadtxt('test.csv', delimiter=',')
rY = data[:, 0]
test = transpose(data[:, 1:data.shape[1]])
test = test[index[:num_feat],:]
```

The last step involves the prediction of the class labels for the test data (note that, the prediction is based on the parameters `w` of the classifier). Finally, the `py` variable contains the predicted labels.

```
# predictions
py = 1/(1+exp(-dot(transpose(test),w))) # predictions for the class
```

Note that, the prediction produced by the classifier is not binary (i.e., $0-1$ values), but in the range $[0,1]$. For this reason, we use 100 different threshold values (from 0 to 1 with step 0.01) to assign class labels (if the prediction is less that the threshold, then we assign the first label). Then, for each case we compute the precision and recall and we plot the corresponding curve (see next paragraph for more details on the evaluation).

[2]Wikipedia's lemma for *logit function*: `http://en.wikipedia.org/wiki/Logit`.

3.3 How to evaluate the results?

In order to evaluate the feature selection methods, we apply the *precision* and *recall* metrics. The precision and recall are given by the following formulas:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \qquad \text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}.$$

Additionally, we compute the precision-recall curve. Note that, the higher the area under this curve, the better the model. In order to compute the area under curve, we apply the trapezoidal rule.

3.4 Tasks to be done

After studying each part of the code and understand the basic functionality, we will examine the following points:

- Initially, run the classifier without feature selection. Observe the precision-recall curve (and the area under curve), as well as the running time.
- Apply feature selection using the χ^2 measure. For different number of features (e.g., $20, 40, 60, 80, 100, 150$) examine the precision-recall curve (and the area under curve), as well as the running time. Does feature selection improve the accuracy? What about the execution time?
- Repeat the above step using the Information Gain measure.
- How correlated is the ranking of the features created by χ^2 and Information Gain? Hint: use the *Kendall tau*[3] rank correlation coefficient for determining the similarity of the orderings of the features, as produced by the different feature selection methods. You can use the build-in function of Python `scipy.stats.kendalltau()`.
- Create a figure (plot) of the area under curve versus the number of selected features. Repeat the same for the running time. What do you observe?

References

[1] Isabelle Guyon, and Andr Elisseeff. "An introduction to variable and feature selection". The Journal of Machine Learning Research 3 (2003): 1157-1182.

[2] Manoranjan Dash, and Huan Liu. "Feature selection for classification". Intelligent data analysis 1.3 (1997): 131-156.

[3] Wikipedia's article for feature selection (December 2015). `http://en.wikipedia.org/wiki/Feature_selection`.

[3] `http://en.wikipedia.org/wiki/Kendall_tau_rank_correlation_coefficient`