Classification: practical issues

Assignment 4

In this lab we will use two types data to perform classification tasks. The first dataset has been adapted from data provided by Wisconsin University, where computer derived nuclear features are used to distinguish malignant from benign breast cytology (that is, breast cancer). Features are computed from digitized images of a fine needle aspirate (FNA) of a breast mass: they describe characteristics of the cell nuclei present in the image. The original images were annotated by an expert with labels M and B. The second dataset will be constructed starting from a set of news documents from the BBC, already used in the previous assignment (acknowledgments in the data folder). The objective of this lab is to address some typical problems related to the practical usage of classification algorithms. You will start with a naïve classification approach, just applying a data mining algorithm on the original data, and see the accuracy you would obtain in this case. To do this you will have to set up a model evaluation process. Then, your objective is to apply the knowledge learned in the first part of the course, and partially applied during the first assignment, to improve this baseline accuracy as much as you can. For your information, the scientific literature reports a best accuracy of 97.5% on these data, computed with repeated 10-fold cross-validations. The assignment concludes with a classification of non-tabular data, where you can test your ability to apply general classification concepts to different types of data.

Task 0: Warm up

Read the cancer.csv data and have a quick look at them. (We assume that) you are not an oncologist, so you are not required to understand the meaning of all attributes: the objective of this assignment is to use automated methods to identify if some of them are useful to classify the data. Of course if this were not a training exercise you would work in collaboration with domain experts. However, some of the names should be easily understandable. Attributes prefixed by "LARGE" indicate the largest values over all cells in the original image, and attributes prefixed by "SE" indicate the standard deviation of that attribute's values over all cells in the original image. You can start by using pandas read_csv() function.You should be familiar with data importing from previous assignments.

Hint: You can use following code to select subset of columns.

```
X, y = dataframe.iloc[:, 2:], dataframe.iloc[:, 1]
```

Task 1: Basic k-NN classification

First import functions that are going to be used in this task using following code.

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
```

To compute the accuracy run a simple k-NN classifier on the original data. To create holdout dataset using in Scikit-learn function:

```
train_test_split(X,y, test_size=0.3, random_state=0)
```

this fucntion returns training data, training labels, test data and test labels. For classifier use:

```
KNeighborsClassifier(n_neighbors, metric="euclidean")
```

In order to get accuracy of the model use:

```
model.score(X=X_testset,y=y_testset)
```

For now use k-NN with k=1 and Euclidean distance as the numerical distance (as you have seen, all regular attributes are numerical) in the training phase. Try to execute it and see if the classification result is satisfactory. You can now try different ways of producing test data. In particular:

- Holdout, 70/30 (called Split Validation in Rapid Miner: the one you just used).
- 10-fold cross validation. Use the sampling method you prefer (e.g., shuffled, building the folds uniformly at random) but please try to relate the method to your knowledge and motivate your choice.
- Leave-one-out.

Useful functions are:

```
scores = cross_val_score(model, X_testset, y_testset, cv=10)
```

Or

```
loo = LeaveOneOut()
# Train model
scores = cross_val_score(model, X, y, cv=loo)
```

Use `cross_val_score` to run k-fold cross-validation where number of folds or data set enumerator (such as leave-one-out) is specified as **cv** argument. To get an idea of the potential impact of K on these results, choose one of the validation approaches (e.g., cross validation) and set K to 10 in K-NN model. Check the change in accuracy. (We will get back to how to find a good value of K later.) **Important:** Sometimes passing pandas dataframe to the classifier might cause an error in newest version of sklearn. Thus instead of passing X use X.values, where X is dataframe or its fragment.

Task 2: Data scaling

One potential problem with k-NN is the presence of significantly different scales in the different attributes. You should now know how to rescale the data, so we do not provide additional details here: what we want to focus on is the impact of preprocessing on classification. Check if this improves accuracy, and also have a look at the rescaled data.

Task 3: Feature selection

Now, try a different approach. The data might contain some attributes that are not useful for the classification, but can make the distance computation less accurate. Instead of rescaling the attributes, try to select a small subset of them. You can do it using the Forward Selection algorithm. In forward selection For each added attribute, the performance is estimated using some validation operator (in this case just use the 10-fold cross-validation, that you can copy and paste into the Forward Selection sub-process). Only the attribute giving the highest increase in performance is added to the selection. Then a new round is started.
Firstly let's import used functions

```
from sklearn.feature_selection import SequentialFeatureSelector
```

Then forward selector can be run using following function:

```
selector=SequentialFeatureSelector(knn_model, n_features_to_select=10 ,direction="forward")
```

Attribute selection is accessed by:

```
selector = selector.fit(X.values, y)
selector.get_support()
```

where `get_support()` returns list of boolean values where, `True` indicates selected attribute. After obtaining attribute selection we can use:

```
X = selector.transform(X)
```

to apply selector results and drop less useful attributes.

Task 4: Combining the two methods

Now that you have tried the practical potential impact of these approaches, combine them together to see if their combination can further improve your results: first rescale the attributes, then apply forward selection to the normalized data.

Task 5: Principal Component Analysis

Now, you can try PCA as an alternative approach to reduce the dimensionality of the data. In summary, please replace the Forward selection with a PCA transformation. First, have a look at the result of the PCA transformation, check the variance captured by each of the components. Use following code:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=5)
X = pca.fit_transform(X)
pca.explained_variance_ratio_
```

Then, use the transformed data for classification. Reduce their dimensionality keeping only some of the components. Try with 1, 5, 10, 15, 20, and check the corresponding accuracy. Also double-check what happens to the accuracy if you invert Normalize and PCA, that is, you first compute the principal components and then normalize them. Before doing it: what would you expect? Was the effect the expected one?

Task 6: Optimizing the parameters – K

In the previous task, you had to check different parameters for the "number of components". This operation can be automated. We will try this here for the K parameter of the k-NN classifier: how can we choose the best K (that is, the one leading to the best accuracy computed on the test data)? In order to optimize k hyper-parameter we can use grid search. Grid search can be used optimized whole set of parameters at the same time, however in this assignment it is used just to optimize one is that K. Look at the following code:

```
parameters = {'n_neighbors': [i for i in range(1,20)]}
knn_model = KNeighborsClassifier()
classifier = GridSearchCV(knn_model, parameters)
classifier.fit(X.values,y)
```

Above code is testing `n_neighbors` values from 1 to 20. `classifier.cv_results_` contains results of the search. We can use following snippet to determine best value for number of neighbors:

```
best_idx = np.argmax(classifier.cv_results_['mean_test_score'])
classifier.cv_results_['params'][best_idx]
```

After executing the process, in the Results panel you can look at the accuracies obtained using different values of K. Use the Charts view to plot them. For now, we only optimize for K. You are free to do the same for other parameters as well, for example the number of components kept after PCA. This is not required for this assignment though.

Task 7: Decision tree

The analysis you have performed so far has told you something about the potential issues in the original data: scale and attribute relevance. This suggests that you should also try using an approach that – as you know – is less sensitive to these problems: decision trees. As decision trees are not affected by scaling problems and can to some extent autonomously perform the selection of relevant features. In order to use decision tree first import it from sklearn package:

```
from sklearn import tree
```

then to initilize it, use:

```
decision_tree = tree.DecisionTreeClassifier()
```

This returns decision tree classifier which can be used similarly to K-NN classifier.

You can now simplify the process and validate directly on the original data, without standardization nor PCA. Check the impact on the accuracy. Have a look at the generated model (the tree). In order to visualize tree model use graphviz package, which first needs to be installed using:

```
pip install graphviz
```

or

```
conda install -c anaconda graphviz
```

after installation, import it and call on the trained tree to save its visualization in .pdf file:

```
import graphviz
data = tree.export_graphviz(decision_tree, out_file=None)
graph = graphviz.Source(data)
graph.render("file_name")
```

Try to understand what discriminates between a benign and a malignant case. As a final change, set the argument of the tree `min_samples_leaf` to 20. This should result in a more compact tree. How many nodes does it contain? How does it relate to the previous tree you obtained?

Task 8: Text classification

This task is intended to be performed independently, so we only provide limited instructions. The knowledge acquired during the lectures and from the assignments should be sufficient for you to set up this process from the beginning to the end. To perform this task you need to use `sklearn.feature_extraction.text` and `nltk` package or `gensim` package. You will use the data bbc.zip, containing five folders (business, tech, sport, ...), each containing several news articles about the same topic of the folder. The objective of the task is to set up and validate a classifier to identify the topic of the articles based on their text. We recommend to try different options, so that you can reflect about their impact (or absence of impact) on this classification task. To make it easier for you to start we provide a following code that loads data from the directory:

```
from sklearn.datasets import load_files
dataset = load_files('dataset_folder/bbc/bbc',encoding='latin-1')
X, y = dataset.data, dataset.target
```

Other usefull functions that you can look into are:

```
nltk.tokenize.word_tokenize
nltk.corpus.stopwords
```

In order to use nltk tokenizer you first need to download wordlist using `nltk.download('punkt')`. If you want to use nltk stopwords you need to first download manually stopwords list, running following command in terminal `python -m nltk.downloader stopwords`. Some of the text vectorization functions are available in `sklearn.feature_extraction.text`.

Task 9: Optional

In this assignment we have used only two types of classifiers, to test the impact of different choices on the result. You (may) have learned other types of classifiers in other courses, or be interested in trying others. You are of course free and welcome to try any other model algorithm available in sklearn package or other packages, including torch or keras.