Machine Learning in Python

Martine De Cock

1 Getting Started with Python

- In this course we use Python 3. If not already done so, please install it on your system. If you are new to Python, the easiest way to get started is to install Anaconda with Python 3. This installation includes all the popular data science packages that you will need for this course. Simply go to the website, download, and install (choose Python 3): `https://www.continuum.io/downloads`
- A useful reference book on Python is: "The Quick Python Book", 2nd edition, Naomi R. Ceder, Manning Publications, 2013.
- Many other reference materials and tutorials are available online. Among these, "The Hitchhikers Guide to Python!" is a popular and well known resource: `http://docs.python-guide.org/en/latest/`

Files needed for this exercise: YouTubeNB.py, YouTube-User-Text-gender-personality.tsv

Exercise 1.1 Text classification using Naive Bayes. This problem is about predicting the gender of vloggers from the text transcripts of their vlog. A video blog or video log, usually abbreviated as vlog, is the video form of a blog. Vloggers explicitly show themselves in front of a webcam, talking about a variety of topics including personal issues, politics, movies, books, etc. The dataset provided for this problem contains text transcripts of 404 vlogs, as well as the gender of the vloggers and their Big Five personality scores as perceived by annotators watching the vlogs. In this problem we will *only use the text transcripts and the gender*.

Get the code YouTubeNB.py and the data file YouTube-User-Text-gender-personality.tsv from the course Canvas website, in the folder Files/python-practice. Open the file that contains the code and inspect it. Notice how the data from the tsv-file is read into a pandas dataframe:

```
pd.read_table("YouTube-User-Text-gender-personality.tsv")
```

Next, the data is split into 300 training examples, and 104 test examples. The training examples are used to build a Naive Bayes classifier for gender recognition from the transcripts. Next this classifier is applied to the transcripts of the 104 test examples. We will discuss Naive Bayes in detail later in the course. We will also look at more examples of training and testing machine learning models in Python, and there will be exercises that prompt you to build and test models yourself.

For now, please run the code, write down the accuracy result, and bring it to class. Do not share the dataset with anyone outside of the course.

2 Pandas, Scikit-learn, and Linear Regression

Files needed for this section: AdvertisingLR.py, Advertising.csv

Building a machine learning model in Python usually starts with loading a dataset, exploring and preprocessing it. Pandas is a popular Python library that offers many functionalities for this. As is usual for Python packages, we first need to import it:

```
import pandas as pd
```

After reading a csv-file into a Pandas dataframe, we can inspect the first five rows of data by using the `head()` command.

```
table = pd.read_csv('example.csv')
print(table.head())
```

Once the data is in a dataframe, we can begin to explore it, preprocess it, and extract subsets, using powerful single-line Pandas functions. For an overview, see `http://pandas.pydata.org/pandas-docs/version/0.15.2/tutorials.html`

For building the actual machine learning models, we rely on scikit-learn, a Python library with a great selection of machine learning functions. Scikit-learn is used to build machine learning models, make predictions and compute metrics.

1. Scikit-learn functions take as input NumPy arrays. Pandas is built on top of NumPy.
2. Training a model with scikit-learn is done using the `fit()` function. In the code snippet below, `X_train` is a Pandas dataframe with the values of the input features, and `y_train` is a Pandas series with the values of the response variable, i.e. the output feature:

   ```
   from sklearn.linear_model import LinearRegression
   reg = LinearRegression()
   reg.fit(X_train, y_train)
   ```

3. A trained model can be used to make predictions with the `predict()` function:

   ```
   y_pred = reg.predict(X_test)
   ```

4. The predictive performance of trained models can be evaluated using a variety of functions from `sklearn.metrics` (for example, MAE):

   ```
   from sklearn import metrics
   metrics.mean_absolute_error(y_test, y_pred)
   ```

File needed for this exercise: Facebook-User-LIWC-personality.csv

Exercise 2.1 This problem is about inferring the personality trait scores of Facebook users, given the LIWC features that were extracted from their status updates. The folder Files/python-practice/exercises on the course Canvas website contains a dataset with the following information for each user (see Facebook-User-LIWC-personality.csv):

- The userId.

- 82 LIWC features extracted from the Facebook status updates of the user.[1]
- The 5 Big Five personality trait scores of the user, as scores in the range $[1, 5]$.

Split the data in 8000 training examples, and 1500 test examples. Build a linear regression model for the extraversion personality trait, using the 8000 training examples only. Compute the RMSE of the linear regression model on the 1500 test examples.

Hint: there are many different ways in which you can process lists in Python. The code below might come in handy to create a list of the LIWC features. Use of this code is optional. You are free to use any approach that works for you.

```
df = pd.read_csv('Facebook-User-LIWC-personality.csv', index_col=0)
big5 = ['ope','ext','con','agr','neu']
LIWC_features = [x for x in df.columns.tolist()[:] if not x in big5]
```

3 Decision Trees

Files needed for this section: HeartDT.py, Heart.csv

The scikit-learn library provides functions for training and testing many other kinds of machine learning models, beyond linear regression. The general approach to train a model and then make predictions with it, is entirely the same. Training and using a decision tree can be done for instance as follows:

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_prob = clf.predict_proba(X_test)
```

The difference between the last two lines above is that `y_pred` will contain the class labels, while `y_prob` will contain associated probabilities, i.e. how likely each label is.

File needed for this exercise: BreastCancer.csv

Exercise 3.1 In this exercise we will use the dataset BreastCancer.csv that you can find in Files/python-practice/exercises on the course Canvas webpage. This dataset is a modified version of the well known Wisconsin breast cancer dataset that can be found in the UCI machine learning repository[2]. The input features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. The response variable 'diagnosis' is 2 for benign, and 4 for malignant. There are 478 instances in the data, out of which 458 are benign and 20 are malignant.

[1] For an overview of what LIWC features are, see the paper text-Tausczik-2010.pdf in Files/project/papers on Canvas. The paper text-Farnadi-2013.pdf in Files/project/papers mentions only 81 LIWC features. This is because in our data the first feature (Seg) has the same value for all users. Hence it can be dropped.

[2] http://mlr.cs.umass.edu/ml/datasets.html

- Build a decision tree that classifies new instances as benign or malignant. Train your decision tree on 70% of the data, and reserve the remaining 30% for testing. Notice that because the dataset is imbalanced, i.e. there are many more benign than malignant cases, you should do a *stratified* split that preserves the same ratio of benign vs. malignant cases in the train as well as in the test data. To achieve this, you can use the following piece of code to indicate that the instances should be split in a stratified manner, based on the values in the `y` column:

  ```
  train_test_split(X,y, test_size=0.3, stratify=y)
  ```

  In this way, 14 of the 20 malignant examples will be moved to the training data, while the other 6 malignant examples will go to the test data.

- Compute the accuracy of the obtained decision tree on the test data. Next, compute the AUC value, with '4' (malignant) as the positive label. Which of these two metrics, i.e. accuracy or AUC, can give a better estimate of the performance of your model? Why?

Hint: you can visualize the decision tree using the code:

```
tree.export_graphviz(clf, out_file='tree.dot', feature_names = features)
```

This will create a file tree.dot. You can copy and paste the contents of this file into an online visualization tool like `http://webgraphviz.com/`.