Programming and Data Science for Biology (PDSB)

Session 9
Spring 2018

Today’s topics

1. Review of last assignment
2. Introduction to Machine Learning
3. Data Science: learning by doing.
4. Hands-on practice

Upcoming topics relate to projects

1. **Today:** Sci-kit learn
2. **Next week:** Bayesian and frequentist statistics (scipy & pymc3)
3. **Two weeks:** matplotlib, toyplot, folium.

Last week's assignment:
The records library

The records library (using requests)

We create a simple library with two Class objects in it.

The Records class gets data from the GBIF API,

The Epochs class organizes data from Records objects into a sorted dataframe, and provides functions for analyzing the data, and reloading existing data sets.

Linter updates.
Did your code run? Did it install? These things can all be easily tested.

Two new things:

Kwargs to enter arbitrary arguments to a function.

Pandas chaining of functions to perform calculations or transform data.

```
import pandas as pd
import numpy as np
import records

# get data
ep = records.Epoch(
     “Pedicularis”,  1975, 2000, 25,
     **{“genusKey”: “3171670”}
     )

# groupby, select, apply
nsp = (
     ep.df.
     .groupby(“stateProvince”)
     .species
     .apply(len)
)
```

Simpson’s diversity index

We added a function for calculating diversity as a measurement of the proportion of individuals that are different species.

To calculate this on our data we will use the common workflow to split, calculate, and then rejoin, as described in the pandas chapter in the Data Science Handbook.

This uses .groupby and .apply

```
import pandas as pd
import numpy as np
import records

# get data
ep = records.Epoch(
     “Pedicularis”,  1975, 2000, 25,
     **{“genusKey”: “3171670”}
     )

# groupby, select, apply
nsp = (
     ep.df.
     .groupby(“stateProvince”)
     .species
     .apply(len)
)
```

records library

We now can easily download data that we can mine for useful information.

Many places in our code we need to check for **empty** or **missing data (nan).** This is easily done using the .empty or .isna attributes.

We'll return to the records library later to analyze metrics of diversity through time and space using these tools. First, though, we need to learn more about applying statistical methods.

```
import pandas as pd
import numpy as np
import records

# get data
ep = records.Epoch(
     "Pedicularis",  1975, 2000, 25,
     **{"genusKey": "3171670"}
     )

# groupby, select, apply
nsp = (
     ep.df.
     .groupby("stateProvince")
     .species
     .apply(len)
)
```

What is machine learning, how and when to use it, and why?

What is machine learning?

Machine learning is about extracting knowledge from data. It aims to combine classical statistical methods (e.g., principal components analysis, ordinary least squares), with computer science to develop algorithms that are *predictive*.

Machine learning *usually* doesn't care about identifying a *true* underlying model, whereas in ecology and evolution that is often our goal. But still, there are many applications where machine learning is very useful.

What is machine learning?

Supervised learning
Machine learning methods work best for problems that involve automating *decision-making* processes by generalizing based on known examples.

Unsupervised learning
These implementation are harder, but also more unique to machine learning. Here only input data is provided and a model is trained to detect useful features of the data.

Examples of supervised machine learning?

1. Identify the zip code from handwritten digits.
2. Determine whether a tumor is benign based on medical images.
3. Detect fraudulent activity in credit card transactions.

In each case we can collect many data points of input/output pairs that we can use to *train* an algorithm to predict outputs from inputs.

Examples of unsupervised machine learning?

1. Identify topics in a set of blog posts.
2. Clustering customers into groups with similar interests.
3. Detecting abnormal access patterns to a website.

In each case we can collect many data points of input/output pairs that we can use to *train* an algorithm to predict outputs from inputs.

Preparing data

One of the most important and difficult tasks of machine-learning is preparing your data in the proper format,: a table-like form.

Using numpy:

+ Each data point is a row *(sample)*
+ Each property that describes that data point is in a column *(feature)*

You might have 1000 rows of data describing 1000 customers with columns describing their age, gender, when they last shopped, etc.

Why is it powerful?

1. A particularly powerful feature of machine learning is that once a model has been trained we can apply it make prediction on new data *super fast*.

The new data does not need to be analyzed along with the old data to make predictions.

Why is it powerful?

2. High dimensional data is not problematic, but instead beneficial. In classic statistical problems when data becomes highly dimensional, meaning there are way too many potential explanatory variables, analyses become overburdened. In ML these extra features are simply ignored if they do not provide information, but often *hidden structure* can be found in the data making these features information. In fact, that's the whole point of many unsupervised methods.

The basic workflow

1. Choose a Model class
2. Choose model hyperparameters
3. Fit model to training data
4. Use model to predict labels on test data

\+ Cross validation, leave-one-out, split-test-train, are all methods for optimizing model parameters or hyperparameters based on different ways of sampling the data.

Sidenote: Testing with simulations

Preparing data

One of the most important and difficult tasks of machine-learning is preparing your data in the proper format,: a table-like form.

Using numpy:

+ Each data point is a row *(sample)*
+ Each property that describes that data point is in a column *(feature)*

You might have 1000 rows of data describing 1000 customers with columns describing their age, gender, when they last shopped, etc.

Using simulated data

Learning statistics and statistical methods is hard, takes a long time, and the details are often over our head. The best way to **learn** and **apply** statistical methods is to:

1. Look for applications where others have analyzed similar types of data, or answered similar types of questions, and try to understand their implementations.

2. Simulate data to look like your results under a known generative model, to **validate** that the method is working how you think it is.

Simulating data

1. The numpy random library
2. Wrap as a pandas dataframe

Open notebook
notebooks/nb-9.1-generating-data.ipynb

Validation in machine learning

1. Split data into ‘test’ & ‘training’
2. Train model on ‘training’ data
3. Predict labels on ‘test’ data
4. Perform cross validation to optimize your model hyperparameters, and/or pipeline.

Open notebook
notebooks/nb-9.2-KNN-classifier.ipynb

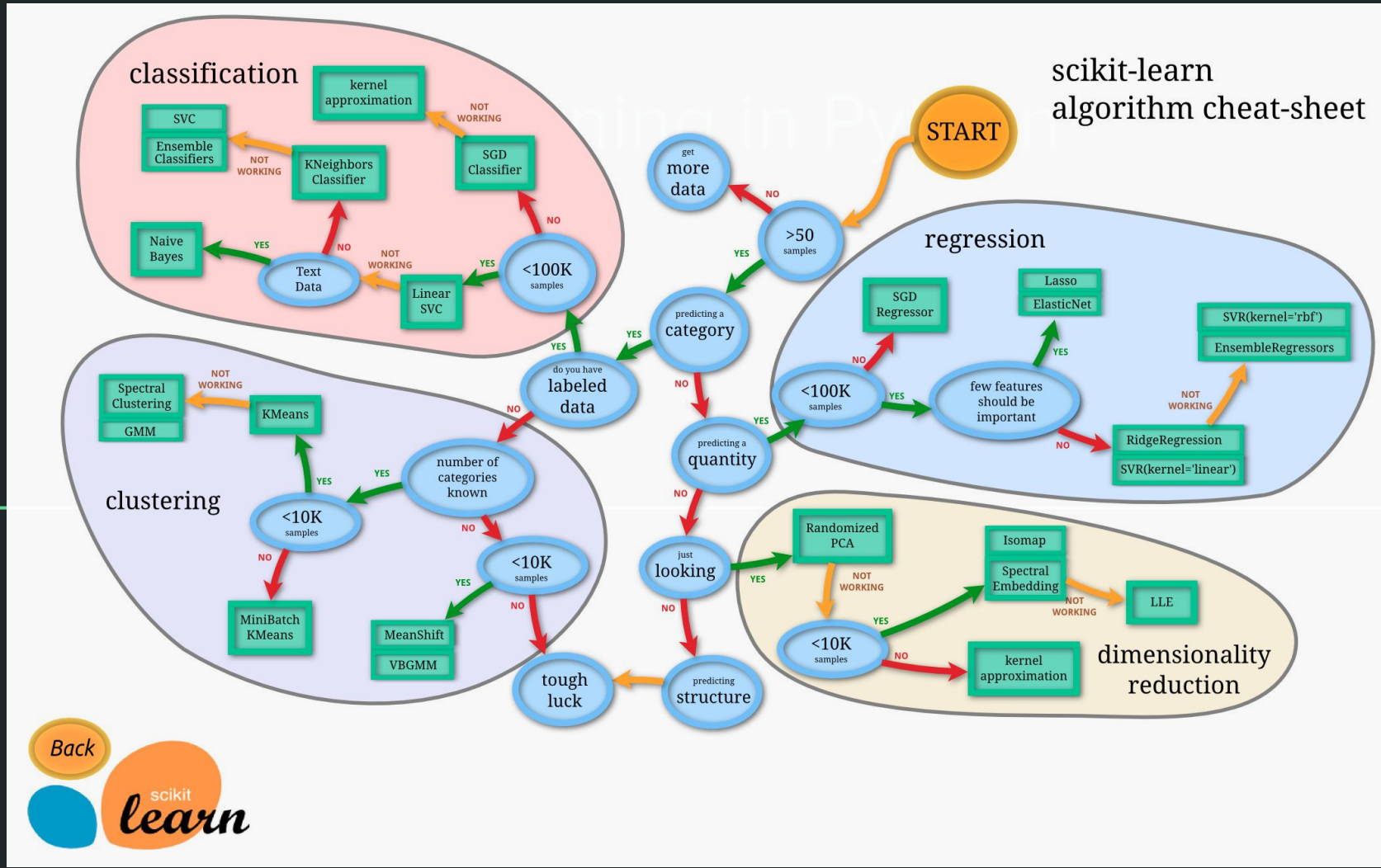
Validation in learning to do machine learning

1. Simulate data with numpy under a *generative process* that could produce the observed data.
2. Examine how well the model infers hyperparameters of the model

Example nb-9.1: I used randomly drawn x values to generate values for y according to a linear regression model, and so we could test how well the model parameters were inferred in the end.

Example nb-9.2: I generated x,y coords from different random distributions, but then fit model using SVM, which tries to draw a line to separate points. It would have been better to have used a generative model that draws a line to assign labels, or to have fit the data with a model that assumes they are drawn from a distribution, such as Naive Bayes.

Machine learning algorithms


scikit-learn
algorithm cheat-sheet
START
get more data
NO
>50 samples
YES
predicting a category
YES
NO
do you have labeled data
YES
NO
predicting a quantity
YES
NO
just looking
YES
NO
predicting structure
tough luck
classification
kernel approximation
NOT WORKING
SVC
Ensemble Classifiers
NOT WORKING
KNeighbors Classifier
SGD Classifier
NO
Naive Bayes
YES
Text Data
NO
NOT WORKING
Linear SVC
YES
<100K samples
YES
clustering
Spectral Clustering
GMM
NOT WORKING
KMeans
YES
number of categories known
YES
NO
<10K samples
NO
MiniBatch KMeans
<10K samples
YES
NO
MeanShift
VBGMM
regression
SGD Regressor
NO
Lasso
ElasticNet
YES
SVR(kernel='rbf')
EnsembleRegressors
<100K samples
YES
few features should be important
NOT WORKING
NO
RidgeRegression
SVR(kernel='linear')
dimensionality reduction
Randomized PCA
YES
NOT WORKING
Isomap
Spectral Embedding
NOT WORKING
LLE
<10K samples
YES
NO
kernel approximation
Back
scikit learn

Read the docs

It can be daunting to look at ML code and try to figure out how they came to choose a particular model, or set of models to fit some data. The best way to learn is to run practice examples.

The scikit-learn tutorial is incredibly good for this. Most examples use either prepared data sets (e.g., Iris) or generate data sets, and the results are plotted as well for visual confirmation.

Why Python?

Very good machine learning libraries have been developed in Python specifically because Python is easy to use:

+ scikit-learn
+ tensorflow
+ keras

Easy to write and share algorithms. Actual routines are compiled code (like in numpy). Easy to deploy in cloud, and on large CPU or GPU clusters.

Why Python?

Python also has many nice tools for analyzing image and video data, which is a relatively new type of analysis.

+ OpenCV
+ scikit-learn
+ scikit-image
+ matplotlib

Which: Scikit, keras, tensorflow?

Scikit-learn is the easiest to use and learn, and by-far has the best documentation.

The other two libraries are more optimized for speed and for super large data sets. They focus specifically on maximizing computation using graphic computing power (GPUs).

As you are getting started I recommend first learning with scikit-learn and transferring later to another library once you've mastered the concepts.

An assignment will be posted by tomorrow.
See assigned reading in syllabus:

Assignment: Link to Session 9 repo
Readings: See syllabus
Collaborate: Work together in this gitter chatroom