COMP 364: Computer Tools for Life Sciences

Intro to machine learning with scikit-learn

Christopher J.F. Cameron and Carlos G. Oliver

Key course information

Assignment #4

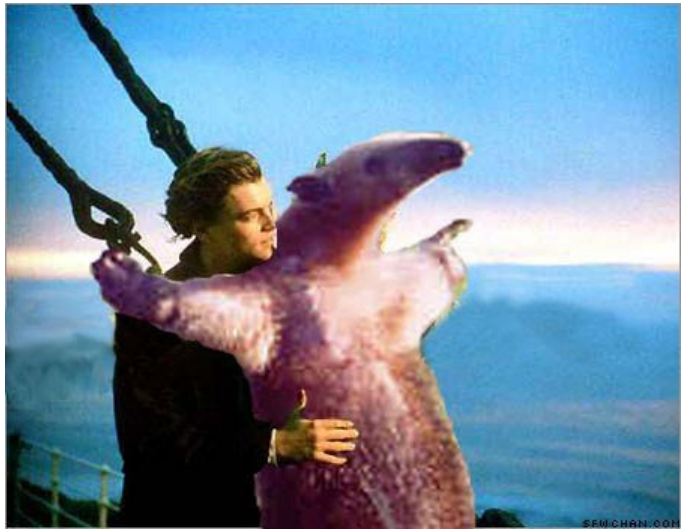
- available now
- due Monday, November 27th at 11:59:59 pm
- first two parts can be completed now
  - remaining concepts will be taught today and on Monday

Course evaluations

- available now at the following link:
  - `https://horizon.mcgill.ca/pban1/twbkwbis.P_WWWLogin?ret_code=f`

Problem: predicting who will live or die on the Titanic

Passenger survival data
`http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic3.xls`



To read in the Excel '.xls' file, we will use the Pandas Python module

- **API**
  `http://pandas.pydata.org/pandas-docs/stable/`
- **tutotials**
  `https://pandas.pydata.org/pandas-docs/stable/tutorials.html`

Parsing an Excel '.xls' file with Pandas

```
1  import pandas as pd
2
3  # parse Excel '.xls' file
4  xls = pd.ExcelFile("./titanic3.xls")
5  # extract first sheet in Excel file
6  sheet_1 = xls.parse(0)
7  # get list of column names
8  print(list(sheet_1))
9  # prints: ['pclass', 'survived', 'name',
10 #  'sex', 'age', 'sibsp', 'parch', 'ticket',
11 #  'fare', 'cabin', 'embarked', 'boat', 'body',
12 #  'home.dest']
```

Passenger survival data

In the 'titanic3.xls' file:

- each row is a passenger
- each column is a **feature** describing the current passenger
- there are 14 features available in the dataset
- For example, the first passenger would be described as:

  Miss. Elisabeth Walton Allen (female - 29)

  A first class passenger staying in cabin B5 with no relatives on board that payed $211.3375 for ticket number #24160. She came aboard at the Southampton port to arrive at St Louis, MO. Mrs. Allen survived the titanic incident and was found on lifeboat #2.

Available dataset features

1. 'pclass' - passenger class (1 = first; 2 = second; 3 = third)
2. 'survived' - yes (1) or no (0)
3. 'name' - name of passenger (string)
4. 'sex' - sex of passenger (string - 'male' or 'female')
5. 'age' - age of passenger in years (float)
6. 'sibsp' - number of siblings/spouses aboard (integer)
7. 'parch' - number of parents/children aboard (integer)
8. 'ticket' - passenger ticket number (alphanumeric)
9. 'fare' - fare paid for ticket (float)
10. 'cabin' - cabin number (alphanumeric - e.g. 'B5')
11. 'embarked': port of embarkation
    (C = Cherbourg; Q = Queenstown; S = Southampton)
12. 'boat' - lifeboat number (if survived - integer)
13. 'body' - body number
    (if did not survive and body was recovered - integer)
14. 'home.dest' - home destination (string)

Data mining

Determining passenger survival rate

```
1  from collections import Counter
2
3  # count passengers that survived
4  counter = Counter(sheet_1["survived"].values)
5  print(counter) # prints 'Counter({0: 809, 1: 500})'
6  print("survived:",counter[1]) # prints: 'survived: 500'
7  print("survival rate:",counter[1]/(counter[1]+counter[0]))
8  # prints 'survival rate: 0.3819709702062643'
```

Data mining #2

There are some obvious indicators of passenger survival in the data

```
# get the number of passengers with a body tag
# and their survival status
counter = Counter(sheet_1.loc[sheet_1["body"].notna(),
                              "survived"].values)
print(counter) # prints: 'Counter({0: 121})''
```

It appears that anyone with a body number did not survive

- this feature would be accurate at determining survival
- but, it's not too useful
  - i.e., the passenger would need to already be dead to have a number

Data mining #3

We could also look at how mean survival is affected by another feature's value

For example, passenger class:

```
print(sheet_1.groupby("pclass")["survived"].mean())
# prints:
# pclass
# 1    0.619195
# 2    0.429603
# 3    0.255289
# Name: survived, dtype: float64
```

From the mean survival rates

- first class passengers had the highest chance of surviving
- survival rates correlates nicely with passenger class

Data mining #4

With Pandas, you can also group by multiple features

For example, passenger class and sex

```
print(sheet_1.groupby(["pclass","sex"])["survived"]
      .mean())
# prints:
# pclass  sex
# 1       female    0.965278
#         male      0.340782
# 2       female    0.886792
#         male      0.146199
# 3       female    0.490741
#         male      0.152130
# Name: survived, dtype: float64
```

As a male grad student, I probably wouldn't have made it...

Why machine learning?

From basic data analysis, we can conclude

- Titanic officers followed maritime tradition
  - 'women and children first'
- if we examined the data more, we would see females
  - were on average younger than male passengers
  - paid more for their tickets
  - were more likely to travel with families

Let's now say that we wanted to determine our own survival

- we could write a long Python script to calculate survival
- but this would be tedious (lots of conditional statements)
- and would be dependent on our knowledge of the data

Instead, let's have the computer learn how to predict survival

Data preparation

Before we provide data to a **machine learning (ML)** algorithm

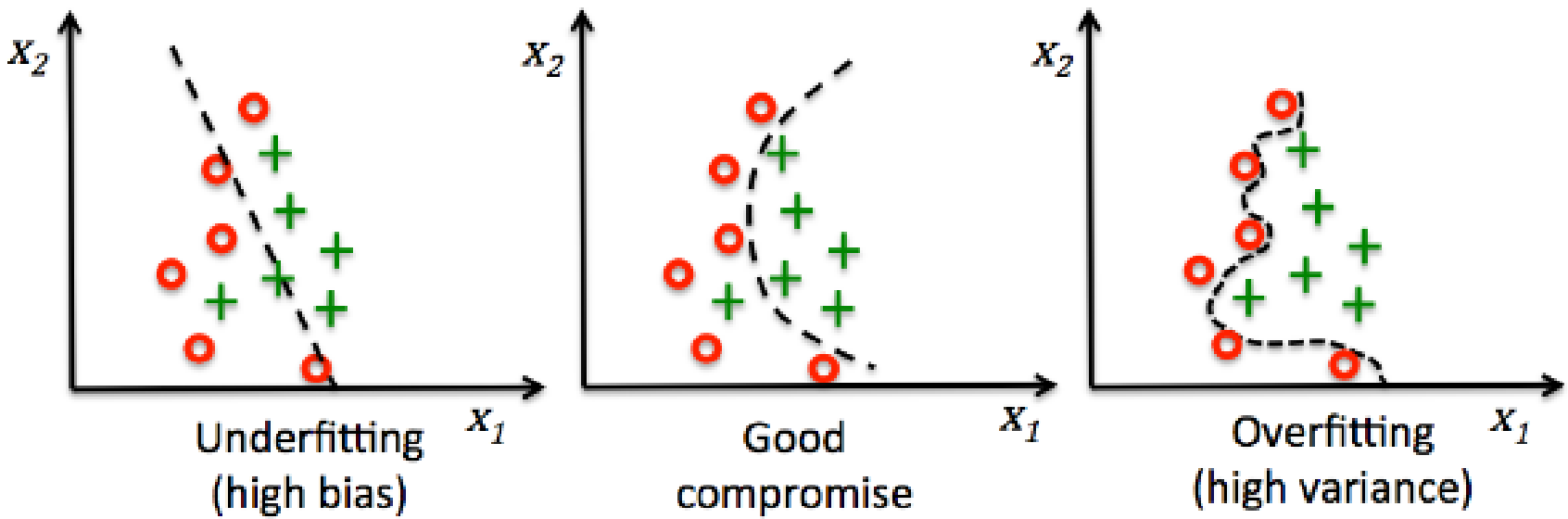
1. remove **examples** (passengers) with missing data
   - some passengers do not have a complete set of features
   - ML algorithms have difficulty with missing data
2. transform features with categorical string values to numeric representations
   - computers have an easier time interpreting numbers
3. remove features with low influence on a ML model's predictions
   - why would we want to limit the amount of features?
   - **overfitting**

Overfitting

What is overfitting?

- occurs when the ML algorithm learns a function that fits too closely to a limited set of data points
- predictions on unseen data will be biased to **training data**
  - increased error for **testing data** during **evaluation**

Example: draw a line that best splits 'o's from '+'s below



Bias vs. variance tradeoff

Bias

- is error from poor assumptions in the ML algorithm
- high bias can cause an algorithm to miss the relevant relations between features and **labels**
- underfitting

Variance

- error from sensitivity to small fluctuations in the training data
- high variance can cause an algorithm to model random noise in training data
  - rather than the intended labels
- overfitting

All ML algorithms work to minimize the error for both the bias and variance

Count the number of examples with a given feature

```
print(sheet_1.count())
# prints:
# pclass        1309
# survived      1309
# name          1309
# sex           1309
# age           1046
# sibsp         1309
# parch         1309
# ticket        1309
# fare          1308
# cabin          295
# embarked      1307
# boat           486
# body           121
# home.dest      745
```

Data preparation #2

Let's drop features with low example counts

- body, cabin, and boat numbers
- home desitnation

```
data = sheet_1.drop(["body","cabin","boat"
                     ,"home.dest"], axis=1)
print(list(data))
# prints: ['pclass', 'survived', 'name', 'sex', 'age',
# 'sibsp', 'parch', 'ticket', 'fare', 'embarked']
```

And remove any examples with missing data

```
data = data.dropna()
```

```
print(data.count())
# prints:
# pclass      1043
# survived    1043
# name        1043
# sex         1043
# age         1043
# sibsp       1043
# parch       1043
# ticket      1043
# fare        1043
# embarked    1043
# dtype: int64
```

Perfect, 1043 examples with a complete feature set

Label encoding

Some of our features are labels, not numeric values

- name, sex, and embarked
- ML algorithms expect numeric values for features

Let's encode them as numeric values

- sex = 0 (female) or 1 (male)
- embarked = 0 (C), 1 (Q), or 2 (S)

Luckily, Python's scikit-learn module has useful methods available

- **scikit-learn API**: `http://scikit-learn.org/stable/modules/classes.html`
- **scikit-learn tutorials**: `http://scikit-learn.org/stable/`

```
from sklearn import preprocessing

le = preprocessing.LabelEncoder()
data.sex = le.fit_transform(data.sex)
data.embarked = le.fit_transform(data.embarked)
print(data[:1])
# prints:
#    pclass  survived                                 name  \
# 0       1         1  Allen, Miss. Elisabeth Walton
#    sex   age  sibsp  parch        ticket      fare  \
# 0    0  29.0      0      0         24160  211.3375
#    embarked
# 0         2
```

Removing unnecessary/misleading features

Unless there is some sick joke to reality

- a passenger's name plays very little importance in their survival

A passenger's ticket number is a mixture of alpha and numeric characters

- it will be difficult to represent as a feature
- may be misleading to the ML algorithm

Like before, we'll remove both from the dataset

Features vs. labels

Now that we have a prepared ML dataset

- split into two lists:
  1. **model input** (or X)
  2. **model targets/input labels** (or y)

```
1 X = data.drop(["survived"], axis=1).values
2 y = data["survived"].values
```

Why should we drop 'survived' from X?

Training vs testing datasets

From the ML dataset select training and testing sets

A ML algorithm will attempt to learn the training dataset

- can be as simple as selecting a random split of data
- 80% for training and 20% for testing
- or may involve more complicated sampling methods

A learned model is not exposed to the test dataset during training

Any predictions on the testing data are designed to be indicative of the performance of the model in general

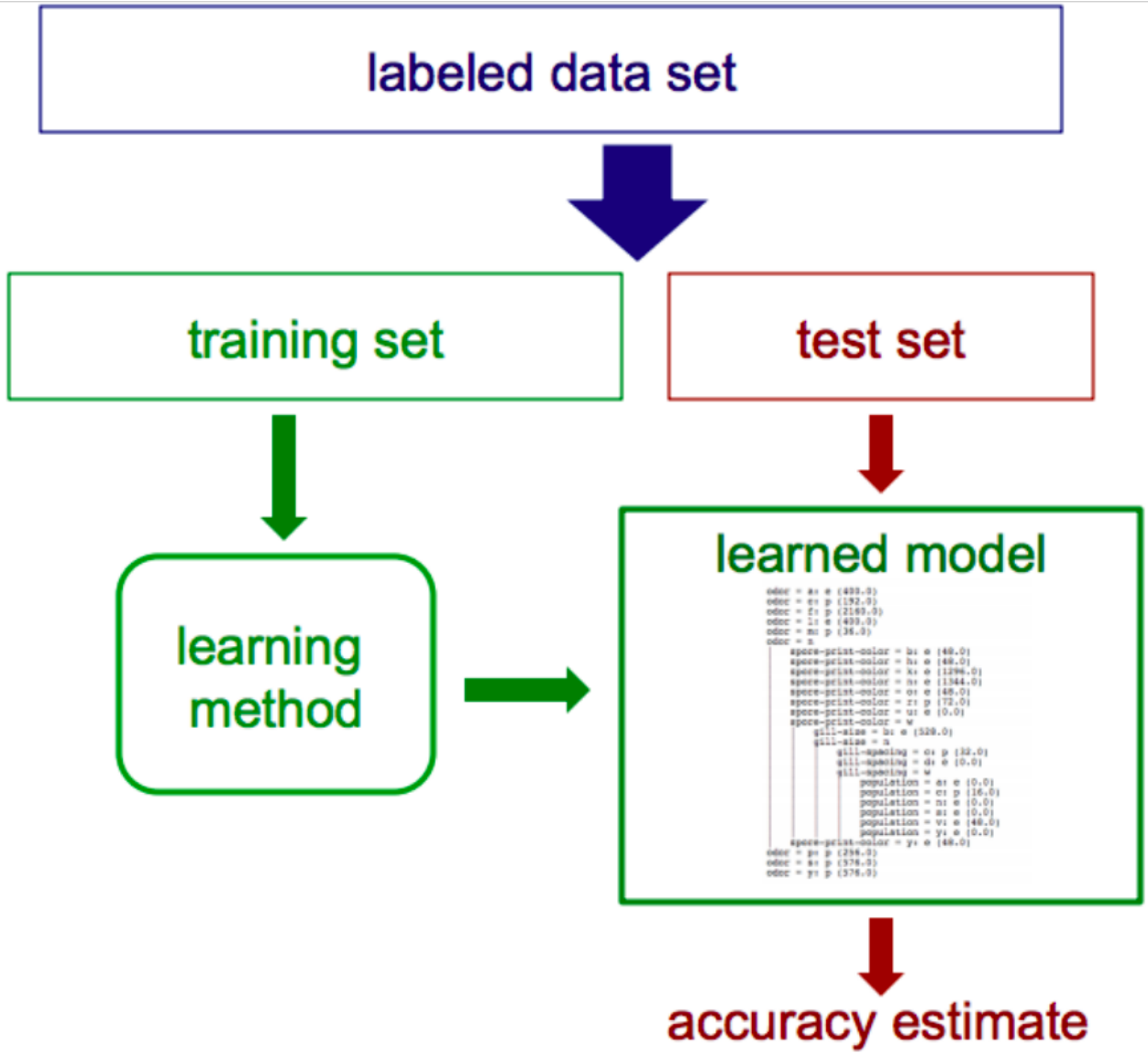
- make sure the selection of your datasets are representative of the problem you are solving
- remember back to 'cat vs. bird', we want pictures of both cats and birds in the training and testing data

Training vs testing datasets #2

In scikit-learn, we can easily create training and testing datasets

```
1 from sklearn import model_selection
2
3 X = data.drop(["survived"], axis=1).values
4 y = data["survived"].values
5 results = model_selection.train_test_split(X, y,
6     test_size = 0.2, shuffle = True)
7 X_train, X_test, y_train, y_test = results
```

Model evaluation



Next time in COMP 364

More of Python's scikit-learn module:

1. ML algorithm selection
   - **support vector machines (SVM)**
2. fitting a function and creating a learned model
   - making predictions using a learned model
3. accuracy estimates
   - **true/false positive (TP/FP) rates**
   - error measures
   - **receiver operating characteristic (ROC)** curves?