xHour Mini Tutorial on Python Programming for Machine Learning

COSC74/174: Machine Learning and Statistical Data Analysis | Winter 2019 | Dartmouth

Outline

In this mini tutorial session, you will learn:

- Python installation and programming interfaces
- Basic python
- The *NumPy* module for mathematical operations
- *Scikit-learn* (*sklearn*) for machine learning

Along the way, we will show examples and/or resources for:

- (Optional) *pandas* and *scipy* modules
- (Optional) Data visualization using *matplotlib*
- (Optional) Writing documents with math expressions using LaTex (Overleaf) & Markdown

Python Versions and Dependencies

(Credit: E. Yao'19, J. Wei'20, D. Chen GR)

Why Python at all?

- Fast to code. Syntax is very easy, many built-in functionalities with the language (`sum()`, `max()`, `min()`, etc.)
- Flexible. If you really want performance, it's pretty easy to write *C* extensions to code to make things run faster. In fact, the *NumPy* library is written in *C*.
- Forgiving. Unless you're working in a large codebase or mission-critical code, why would you ever want static typing?
- Popularity in machine learning, deep learning and data science community

Python3 is recommended. Most of data-science frameworks, particularly more recent ones, are adapted to, or developed in, Python3. It's OK if you prefer to use Python2, just keep in mind some commands may be different.

Programming Interfaces for Python

Jupyter Notebook (recommended)

Pros

- Free and open source
- Supports both code and Markdown chunks that can be assembled into a professionally looking document
- Compatible with most libraries and standard installation approach

Cons

- Takes time to get used to, particularly if you are used to IDE-based programming (e.g. RStudio)

For COSC74/174 Canvas submission

- Execute all code chunks (`Kernel` -> `Restart and Run All`), and save the notebook. Submit the *.ipynb* file
- (Optional) HTML or PDF report of your *.ipynb* notebook
- Submit any required output files (e.g. CSV results, written description/report) wherever appropriate

Integrated Developer Environment (IDE)

You are free to try out and choose an IDE that works for you.

Examples: PyCharm (IDEA), Spyder (via Anaconda)

Pros

- Pretty, customizable programming interface
- *PyCharm*, for example, has similar layout as other products by IDEA (e.g. IntelliJ for Java, Rider for C#)
- Custom programming environment

Cons

- Might not be free or open source (or free with reduced functionality)
- Module installation can be tricky

For COSC74/174 Canvas submission

- Submit *.py* script containing all functions
- Submit any required output files (e.g. CSV results, written description/report) wherever appropriate

Command line (not recommended for beginners)

- Use of your favorite text editor (e.g. *Atom*, *Eclipse*, *VS Code*, *TextWrangler*)
- In a terminal/command line, execute python script via `python3 <script-name>`

The following structure is commonly used:

```
if __name__ == "__main__":
    main()
```

Same Canvas submission procedure as the previous slide

Anaconda Python Distribution

(Credit: E. Yao'19, J. Wei'20, D. Chen)

A distributor of python that bundles many useful components for data science and machine learning!

Pros:

- Choice between Python 2 and Python 3 (e.g. `Kernel` -> `Change Kernel`)
- Many libraries are preinstalled (`numpy`, `scipy`, `django`, `jupyter`, `beautifulsoup`, etc.)
- Also has a package manager like pip! (`conda install <module-name>`)
- Easy to install - Windows and Mac installers are the 'clicky kind'. You can probably get by without opening a Terminal.

Cons:

- Still "semi-global" - you can screw up dependencies in anaconda, but at least you don't have to delete python from your system!

Anaconda installation/usage instructions:

1. Download Anaconda from https://www.anaconda.com/download/#macos (https://www.anaconda.com/download/#macos)
2. Run `jupyter notebook` in Terminal to open up jupyter notebook.
3. To create an environment, put in terminal: `conda create -n <your-env-name> python=x.x anaconda`
4. To activate the environment, `source activate <your-env-name>`
5. To install libraries, run `pip3 install sklearn`

(Optional) Installation via Virtual Environment (*venv*)

(Credit: E. Yao'19)

virtualenv installation/usage instructions

1. Install *virtualenv* using `pip`/`pip3` for an existing python distribution (do this only once) (`pip install virtualenv`, `pip3 install virtualenv` should work for most people)
2. Create a *virtualenv* for an example project inside the folder (do this once per project) (`cd to/project/folder && virtualenv .env` (NOTE: *.env* can be any folder you prefer). *virtualenv* will install a python interpreter and depedencies to the folder you just created
3. Start the *virtualenv* (need to do this every time before you start working on the project). Assuming you installed to *./.env*, run `source .env/bin/activate` or `.env/bin/activate`
4. Once you are done with everything, be sure to run `deactivate` to quit *virtualenv*

If the above works, you should see some informations in parenthesis before the BASH/shell prompt.

Now, whenever you `pip install` something, it will be installed to the subfolder *.env* and you will only be able to use it when you have the virtualenv activated. While venv is activated, your global python installation won't be touched.

To remove a *venv*, you can simply delete the *.env* folder.

Basic Python Tutorial

These are tutorials that you can learn and practice at your own times.

- Variables
- Lists (arrays)
- Dictionaries
- Control structure
- Functions

Variables, Types, and Basic Math

In [1]:
```
a = 2
b = 3
print(type(a), type(b))
```

```
(<type 'int'>, <type 'int'>)
```

In [2]:
```
c = a + b
ab = a * b

print ("Sum: " + str(c))
print ("Product: " + str(ab))
```

```
Sum: 5
Product: 6
```

Special operators

```
In [3]: a = 2
        a += 1 # a = a + 1
        print("a is now equal to " + str(a))
```

```
a is now equal to 3
```

```
In [4]: b = 3
        b *= 3 #b = b*3
        print("b is now equal to " + str(b))
```

```
b is now equal to 9
```

Lists

- Creation
- Concatenation
- Access elements within lists
- List comprehension

List creation

```
In [5]:
zero_list = [0] * 3
string_list = ['hello'] * 3
empty_list = list()
# empty_list = [] #alternative

print(zero_list)
print(string_list)
print(empty_list)
```

```
[0, 0, 0]
['hello', 'hello', 'hello']
[]
```

List concatenation and access

In [6]:
```
## Create new list:
new_list = zero_list + string_list
print(new_list)
```

```
[0, 0, 0, 'hello', 'hello', 'hello']
```

In [7]:
```
## Access elements in a list (i.e. indexing):
print(new_list[0])
print(new_list[0:2])
print(new_list[-1])
```

```
0
[0, 0]
hello
```

List comprehension

List comprehension is not found in other programming languages, but very common and effective in python.

```
In [8]: ## Example 1:
        lc = [a for a in range(5)]
        print(str(lc))
```

```
[0, 1, 2, 3, 4]
```

```
In [9]: ## Example 2:
        import string
        wordsCapitalized = [string.capwords(name) for name in ['i','love','machine','learn
        ing']]
        print(" ".join(wordsCapitalized))
```

```
I Love Machine Learning
```

Check out some more details with hands-on example in this DataCamp blogpost (https://www.datacamp.com/community/tutorials/python-list-comprehension).

Dictionaries

Important: Unlike lists that are ordered, dictionaries aren't.

In [10]:

```
empty_dict = dict()
# empty_dict = {} #alternative

## Key-value pair:
## Integer keys:
num_key_dict = {3: 'Hello', 4:'World!'}
print(num_key_dict)
```

```
{3: 'Hello', 4: 'World!'}
```

In [11]:

```
## String keys:
string_key_dict = {'Hello':3, 'World!':4}
print(string_key_dict)
```

```
{'World!': 4, 'Hello': 3}
```

Control Structure

- `if`-`elif`-`else` statements
- `for` loops
- `while` loops

if-elif-else statement

```
In [12]: x = 20

         if x < 0:
             x = 0
             print('x is negative but now changed to zero!')
         elif x == 0:
             print('Zero')
         elif x == 1:
             print('One')
         else:
             print('Something else')
```

```
Something else
```

For loops

Control structure that allows iterative operations (e.g. through a range of values)

In [13]:
```
for i in range(5):
    print('Now on iteration %d' % i)
    print(i**2)
```

```
Now on iteration 0
0
Now on iteration 1
1
Now on iteration 2
4
Now on iteration 3
9
Now on iteration 4
16
```

Note for programmers in C#, PHP: *for* loops in python syntax can accomplish tasks done by *foreach* loop

While loops

Note: You may use the `break` statement to exit loops including a *while* loop

In [14]:
```
counter = 0

while counter <= 10:
    counter += 2
    print(counter)
```

```
2
4
6
8
10
12
```

Functions in Python

Functions using `def` keyword

The `def` keyword is used for "defining" a function in python.

In [15]:
```
def addition(a,b):
    '''
    Doc string (documentation) goes here
    '''
    c = a + b
    return(c)

print(addition(3,4))
```

```
7
```

Note: Key components of your machine learning algorithm implementation are required to be formatted in function format.

(Advanced) Functions using `lambda` keyword

`lambda` is another reserved word in python for more succinctly creating a function.

- `lambda`-based function definition is good for short, simple functions, but less suitable for longer and more complex functions
- A common use of `lambda` is when passing the given function as argument into another function

In [16]:
```
addition_via_lambda = lambda a,b: a+b
print(addition_via_lambda(3,4))
```

```
7
```

The *NumPy* Module

NumPy is one of the most commonly used libraries in python. In a nutshell, *NumPy* supports:

- Basic mathematical operations
- Matrix computations
- Statistical methods (e.g. *t*, Wilcoxon, Kolgomorov-Simirnov tests)
- Data object manipulation (e.g. reshaping)
- Compatibility with operations within built-in python methods and other modules (e.g. *pandas*)

There are numerous free tutorials for *Numpy*, including

- Official documentation (https://docs.scipy.org/doc/numpy/user/quickstart.html)
- DataCamp's Intro to Python for DataScience (https://www.datacamp.com /courses/intro-to-python-for-data-science)
- Udacity's Intro to Data Analysis (https://www.udacity.com/course/intro-to-data-analysis--ud170)
- Harvard EdX's Using Python for Research (https://www.edx.org/course/using-python-for-research)

Array-based operation in NumPy

In [17]:
```
import numpy as np

## Create arrays:
x = np.array([[1,2,3],[4,5,6]], dtype=np.float64)
y = np.array([[7,8,9],[10,11,12]], dtype=np.float64)

print(x)
print(y)
```

```
[[ 1.  2.  3.]
 [ 4.  5.  6.]]
[[  7.   8.   9.]
 [ 10.  11.  12.]]
```

In [18]:

```python
## Element-wise addition:
print(x + y)
# print(np.add(x, y)) #equivalent
```

```
[[  8.  10.  12.]
 [ 14.  16.  18.]]
```

In [19]:

```python
## Element-wise subtraction:
print(x - y)
# print(np.subtract(x, y)) #equivalent
```

```
[[-6. -6. -6.]
 [-6. -6. -6.]]
```

In [20]:
```
## Element-wise multiplication:
print(x * y)
# print(np.multiply(x, y)) #equivalent
```

```
[[  7.  16.  27.]
 [ 40.  55.  72.]]
```

In [21]:
```
## Element-wise division:
print(x / y)
# print(np.divide(x, y)) #equivalent
```

```
[[ 0.14285714  0.25        0.33333333]
 [ 0.4         0.45454545  0.5       ]]
```

In [22]:
```
## Element-wise square root:
print(np.sqrt(x))
# print(x ** (1/2)) #equivalent
```

```
[[ 1.          1.41421356  1.73205081]
 [ 2.          2.23606798  2.44948974]]
```

Summarize data in matrix format

```
In [23]: ## Calculate mean along rows:
         print("Average (arithmetic mean) of rows:")
         np.mean(x, axis=0)

         Average (arithmetic mean) of rows:

Out[23]: array([ 2.5,  3.5,  4.5])
```

```
In [24]: ## Calculate mean along the columns:
         print("Average (arithmetic mean) of columns:")
         np.mean(x, axis=1)

         Average (arithmetic mean) of columns:

Out[24]: array([ 2.,  5.])
```

```
In [25]: ## Calculate matrix mean:
         print("Average (arithmetic mean) of entire matrix x:")
         np.mean(x, axis=None)

         Average (arithmetic mean) of entire matrix x:

Out[25]: 3.5
```

Basic commands to manipulate arrays

In [26]:

```
## Transposing arrays:
print("Original:")
print(x)

print("\n Transposed:")
print(np.transpose(x))
# print(x.T) #equivalent
```

```
Original:
[[ 1.  2.  3.]
 [ 4.  5.  6.]]

 Transposed:
[[ 1.  4.]
 [ 2.  5.]
 [ 3.  6.]]
```

In [27]:

```
## Reshaping arrays:
print("\n Reshaped:")
print(np.reshape(x, (3,2)))
```

```
 Reshaped:
[[ 1.  2.]
 [ 3.  4.]
 [ 5.  6.]]
```

Commands for Matrix Algebra

Please see the numpy linear algebra documentation (https://docs.scipy.org/doc/numpy/reference/routines.linalg.html) for full list of functionality. These commands are very useful especially if you are working with deep learning applications. Below are a few very basic examples.

Dot Product

```
In [28]: print("a dot b is: ")
         np.dot(x[0], x[1])

         a dot b is:

Out[28]: 32.0
```

Cross Product (aka vector product)

```
In [29]: print("a cross b is: ")
         np.cross(x[0], x[1])

         a cross b is:

Out[29]: array([-3.,  6., -3.])
```

Matrix Multiplication

- Be sure to apply the "row-column" (https://en.wikipedia.org/wiki/Matrix_(mathematics)) rule for dimensions. Otherwise you'll see error!
- Remember, order of the two matrices matter!

In [30]:
```
print("Dimension of matrix x:" + str(x.shape))
print("Dimension of mattrix y:" + str(y.shape))
```

```
Dimension of matrix x:(2, 3)
Dimension of mattrix y:(2, 3)
```

The resultant matrix of $XY^\top$ is therefore:

In [31]:
```
np.matmul(x, y.T)
```

Out[31]:
```
array([[  50.,   68.],
       [ 122.,  167.]])
```

Likewise, in this example you can also calculate $\mathbf{X}^\top \mathbf{Y}$:

In [32]:
```
np.matmul(x.T, y)
```

Out[32]:
```
array([[ 47.,  52.,  57.],
       [ 64.,  71.,  78.],
       [ 81.,  90.,  99.]])
```

(Optional) *pandas* module

pandas is an open-source library for efficient data-frame object manipulation in python.

To install *pandas* with Anaconda, run the following in the command line / terminal:

```
conda install pandas
```

or with `pip`:

```
python3 -m pip install --upgrade pandas
```

There are several good tutorials for *pandas*, including:

- Official documentation (https://pandas.pydata.org/pandas-docs/stable/10min.html)
- Free course on Udacity (https://www.udacity.com/course/intro-to-data-analysis--ud170)
- YouTube tutorial by J James (https://www.youtube.com/watch?v=e60ItwlZTKM)

(Optional) *SciPy* and *statsmodel* Modules

SciPy (https://docs.scipy.org/doc/scipy/reference/stats.html) and statsmodel (https://www.statsmodels.org/stable/index.html) provides many useful functions related to scientific computing, particularly *statistics*.

Both modules are compatible with *NumPy* and *pandas*.

You may use these already-implemented functions for this course, and perhaps for your own research and studies as well.

Scikit-learn for Supervised and Unsupervised Learning

You are highly encouraged to review the tutorials in the official documentation (http://scikit-learn.org/stable/tutorial/index.html).

Dataset

We will use the build-in `Iris` data set as an example. Othere toy data sets could be explored (sklearn.datasets (http://scikit-learn.org/stable/datasets) for more detail).

Approaches

- Logistic regression (supervised learning)
- K-means (unsupervised learning)

Supervised Learning: a Simple Example

```
In [33]: import sklearn
         from sklearn.datasets import load_iris #built-in data
         import matplotlib.pyplot as plt #data visualization

         Iris = load_iris()
         X = Iris.data[:,[1,3]] #select 2 features for demo
         y = (Iris.target == 2) #create a binary DV

         print("Dimension of feature space: " + str(X.shape))
         print("Dimension of class label (should be a column vector): " + str(y.shape))
```

```
Dimension of feature space: (150, 2)
Dimension of class label (should be a column vector): (150,)
```

In [34]:
```
## Preview the data:
print(X[1:5,:])
print(y[1:5])

## Proportion of class 0:
print(np.mean(y))
```

```
[[ 3.   0.2]
 [ 3.2  0.2]
 [ 3.1  0.2]
 [ 3.6  0.2]]
[False False False False]
0.333333333333
```

sklearn offers a convenient function, `train_test_split`, to divide you data into a training set and a held-out validation set.

Training-test split is typically done at a ratio, which could be $80 : 20$, $70 : 30$, or even $60 : 40$. There is no steadfast rule on this.

In [35]:
```
from sklearn.model_selection import train_test_split

TEST_SIZE = 0.30 #adjust as desired
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_SIZE, ran
dom_state=42)
```

In [36]:
```python
from sklearn.linear_model import LogisticRegression

## Initialize the logistic regression classifier
clf = LogisticRegression()

## Fit the training set to the initialized classifier
clf.fit(X_train, y_train)
```

Out[36]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

In [37]:
```
## Apply the model to held-out test set:

## Class labels:
test_class = clf.predict(X_test)
print(test_class)

## Probability scores for being in Class 1/True
## Required for computing area under the curve (AUC)
test_score = clf.predict_proba(X_test)
print(test_score[:,1])
```

```
[False False  True False False False False  True  True False  True False
 False False False False  True False False  True False  True False  True
 False  True  True  True False False False False False False False  True
 False False False False  True False False False False]
[  2.03482773e-01   2.54386105e-03   9.38921527e-01   3.82473836e-01
   3.37543332e-01   6.52019113e-03   2.36951279e-01   8.78953518e-01
   6.38984628e-01   2.28876780e-01   6.89320643e-01   4.22750324e-03
   2.82415139e-03   3.64077876e-03   2.54386105e-03   3.24346027e-01
   8.56602972e-01   2.20997742e-01   2.65131755e-01   8.89668882e-01
   4.42209336e-03   6.00259153e-01   6.52019113e-03   8.50961252e-01
   4.74254570e-01   8.94027831e-01   7.60707099e-01   8.62065746e-01
   8.39656062e-03   5.13408067e-03   2.43171893e-03   1.46221422e-03
   2.45220050e-01   3.27970665e-03   4.42209336e-03   8.17837295e-01
   2.83113387e-01   2.82415139e-03   2.43171893e-03   8.14647298e-04
   7.68837219e-01   2.92375811e-01   3.14520960e-01   3.09050622e-03
   2.09370278e-03]
```

Model Evaluation

The confusion matrix is useful in calculating metrics, including

- TPR
- TNR
- FPR
- FNR
- Sensitivity
- Specificity
- Precision
- Recall

See this page (https://en.wikipedia.org/wiki/Confusion_matrix) for detail.

In [38]:
```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, test_class)
```

Out[38]:
```
array([[31,  1],
       [ 1, 12]])
```

The F1 Score (https://en.wikipedia.org/wiki/F1_score) and Area under the receiver-operation curve (AUROC, or ROC) (https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc) are common metrics for evaluate classifiers, and are superior to the "crude" accuracy defined as:

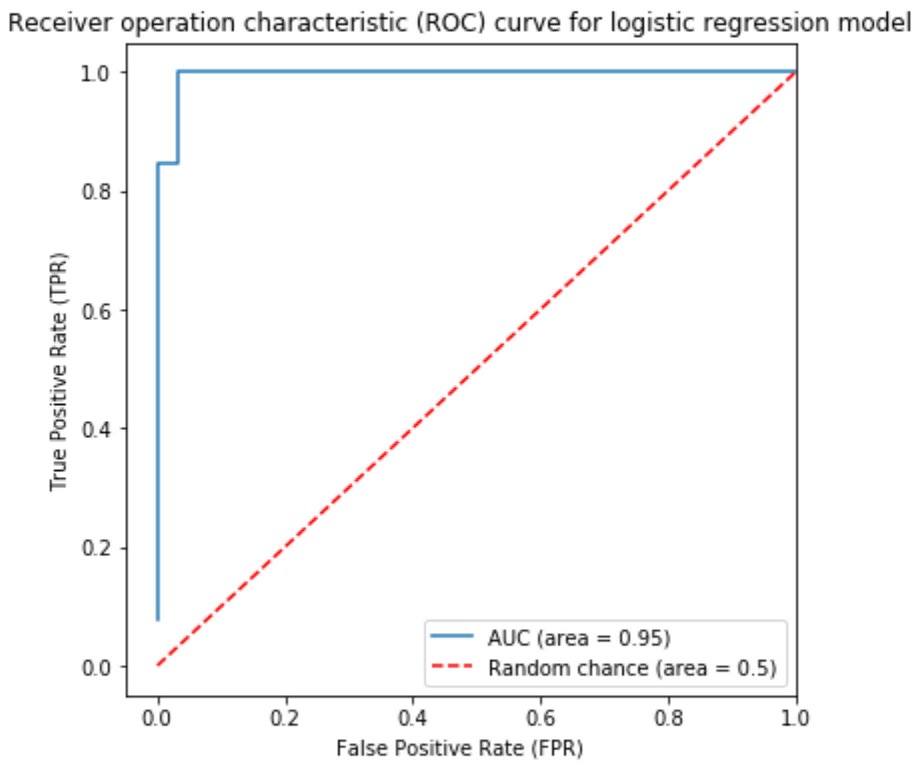
$$accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

Details are to be covered in the course.

In [39]:
```
from sklearn.metrics import f1_score, roc_auc_score, roc_curve

## F1 scores on test set:
f1 = f1_score(y_test, test_class, average="binary", sample_weight=None)
print("F1 score:" + str(f1))

## AUC on test set:
logit_roc_auc = roc_auc_score(y_test, clf.predict(X_test))
print("AUC score:" + str(logit_roc_auc))
```

```
F1 score:0.923076923077
AUC score:0.945913461538
```

```
In [40]: ## Sketch ROC curve and show AUC:
         fpr, tpr, thresholds = roc_curve(y_test, clf.predict_proba(X_test)[:,1])
         plt.figure(2, figsize=(6,6))
         plt.plot(fpr, tpr, label='AUC (area = %0.2f)' % logit_roc_auc)
         plt.plot([0, 1], [0, 1],'r--', label="Random chance (area = 0.5)")
         plt.xlim([-0.05, 1.0])
         plt.ylim([-0.05, 1.05])
         plt.xlabel('False Positive Rate (FPR)')
         plt.ylabel('True Positive Rate (TPR)')
         plt.title('Receiver operation characteristic (ROC) curve for logistic regression m
         odel')
         plt.legend(loc="lower right")
         plt.show()
```



Summary of *sklearn* Supervised Learning

- *sklearn* offers many already written and extensively tested machine-learning algorithms
- Aside of machine learning, *sklearn* offers useful functionalities
- Learning to search and learn from documentation is a critical skill of a good programmer!

Notes on Programming Component of Homework Assignments

- For every machine-learning algorithm, you are required to include the following components as *functions* (though you are free to name your functions differently from below):
  - `train_model(...)`
  - `apply_model(...)` or `predict`
  - `evaluate_model(...)` (optional, doesn't have to be in functional form)
- You may have additional functions (e.g. `cross_validate(...)`), if necessary. Likewise, you may write your own helper functions/methods.
- Good programming etiquette, whcih includes writing clean, well-documented and reasonably commented code, is highly encouraged. Poor organization, messy code shows lack of care and may be penalized.
- While optional, you are encouraged to explore the data before implementing machine learning, visualize the data as well as your results after building the model. These optional procedures do *not* have to be implemented as functions.

(Continued on the next page)

Example: `train_model` function

You could wrap the code shown in the previous several slides as follows.

```
def train_model(X, y):
    '''
    Function to train a logistic regression model showing in the COSC74/174 pytho
n bootcamp
    Parameters
        X: training features
        y: training class labels (0 or 1)
    Returns
        sklearn classifier object
    '''
    clf = LogisticRegression() #initialize
    clf.fit(X, y) #feed in data
    return clf
```

Documentation (doc string) is encouraged but not required.

To use this function:

```
myLogisticModel = train_model(myFeatures, myLabels)
```

Unsupervised Learning: a Simple Example

In unsupervised learning, there is no ground truth, and we determine *clusters* based on some *distance metrics* (more details are to be covered in lecture).

We will use K-means clustering as a demo for unsupervised learning, where K indicates the number of clusters, each of which has a centroid.

In [41]:

```
from sklearn.cluster import KMeans

N_CLUSTER = 2 #a different number of clusters can be used
kmeans = KMeans(n_clusters=N_CLUSTER, random_state=0)
kmeans.fit(X)

cluster_labels = kmeans.labels_
print("Cluster labels:")
print(cluster_labels)
```

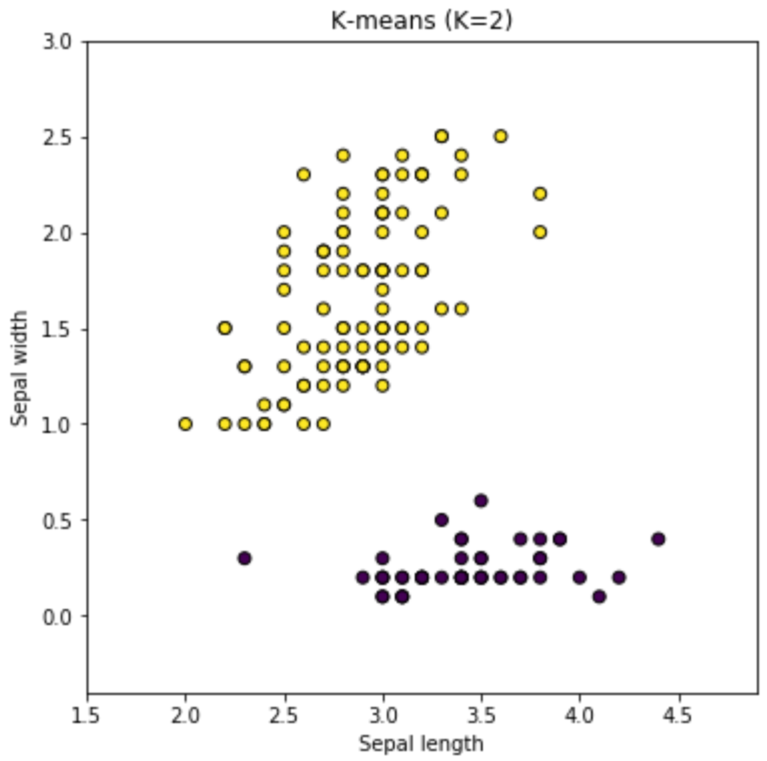
```
Cluster labels:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1]
```

In [42]:

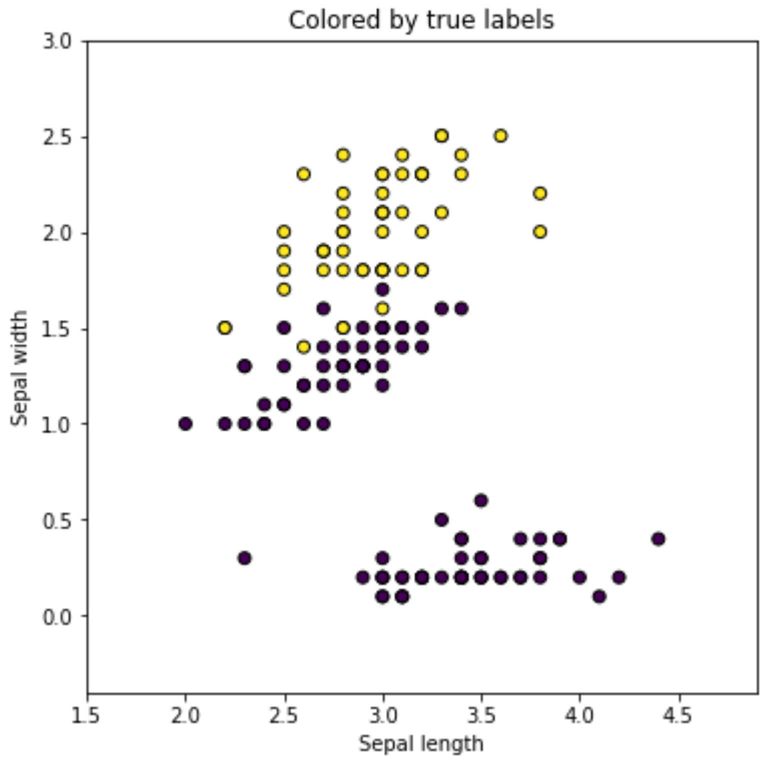
```
## Color by K-means centroids:
x_min,x_max = X[:,0].min()-0.5, X[:, 0].max()+0.5
y_min,y_max = X[:,1].min()-0.5, X[:, 1].max()+0.5
plt.figure(figsize=(6,6))
plt.clf()
plt.scatter(X[:, 0], X[:, 1], c=cluster_labels, edgecolor='k')
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.title('K-means (K=2)')
```

Out[42]:

```
Text(0.5,1,u'K-means (K=2)')
```



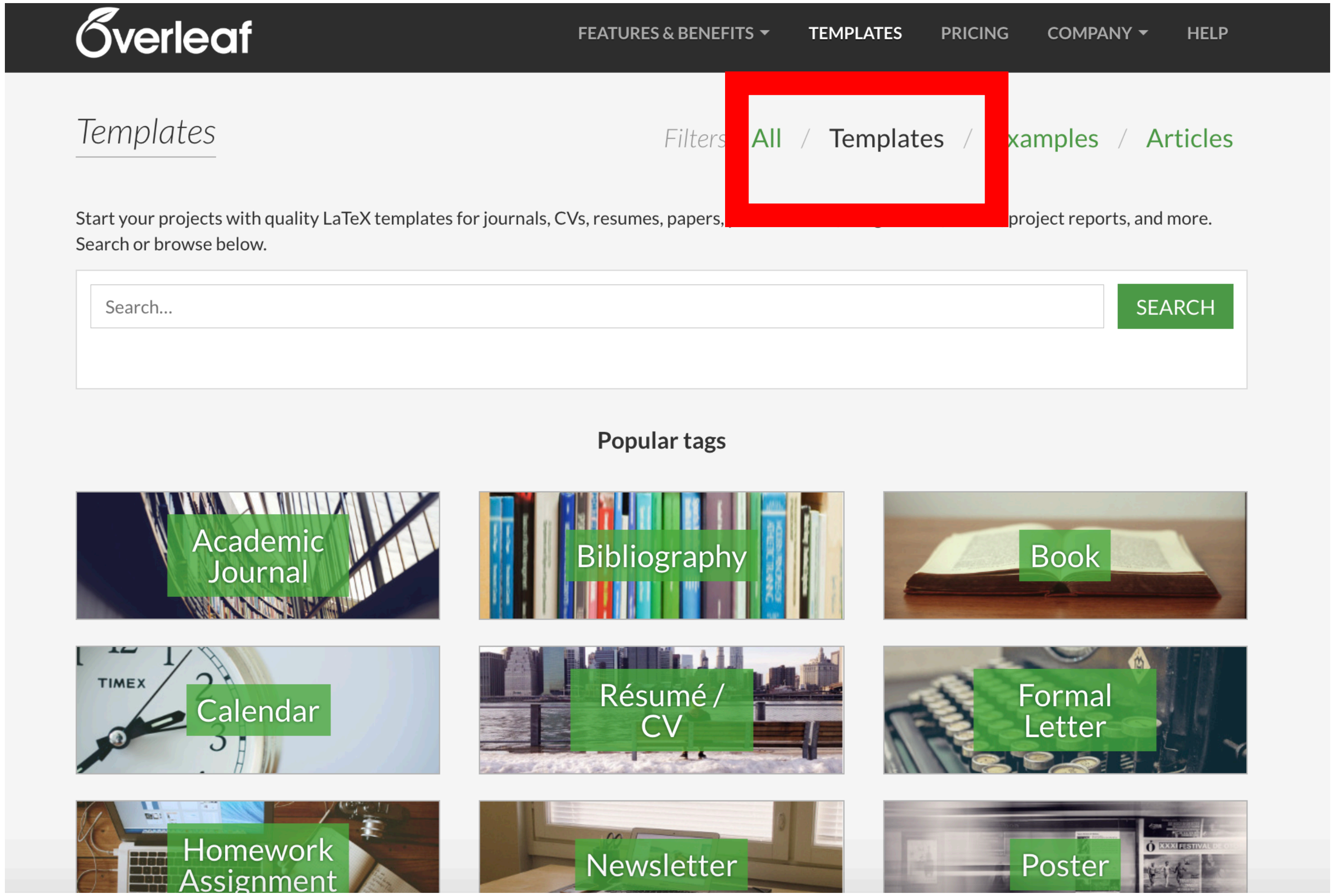
```
In [43]: ## Color by true labels:
         plt.figure(figsize=(6,6))
         plt.clf()
         plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')
         plt.xlabel('Sepal length')
         plt.ylabel('Sepal width')
         plt.xlim(x_min, x_max)
         plt.ylim(y_min, y_max)
         plt.title('Colored by true labels')

Out[43]: Text(0.5,1,u'Colored by true labels')
```



Resources on LaTex/Markdown Language for Writing Report Consisting of Math/Scientific Expressions

- For non-programming, theoretical homework problems, you are highly encouraged to use LaTex to produce a professional-looking document, instead of having them handwritten and scanned/photographed
- In fact, in the programming portion of the presentation, you already saw the use of Markdown/LaTex language.
- If you are using Jupyter Notebook, the “Markdown” option for any given chunk supports LaTex syntax.
- Take a look at this LaTex tutorial (https://www.latex-tutorial.com/quick-start/)
- There are also numerous good YouTube tutorials, such as this one by L. Smith (https://www.youtube.com/watch?v=mfRmmZ_84Mw).
- LaTex can be operated in many editors (e.g. MikTex, TexStudio), which will require installation. Overleaf is an online editor that does not require installation.

Overleaf is a free online Latex platform. You can use it without installing any software and packages.
It also provides you with a lot of templates, not only for your homework, but also thesis paper, CV…

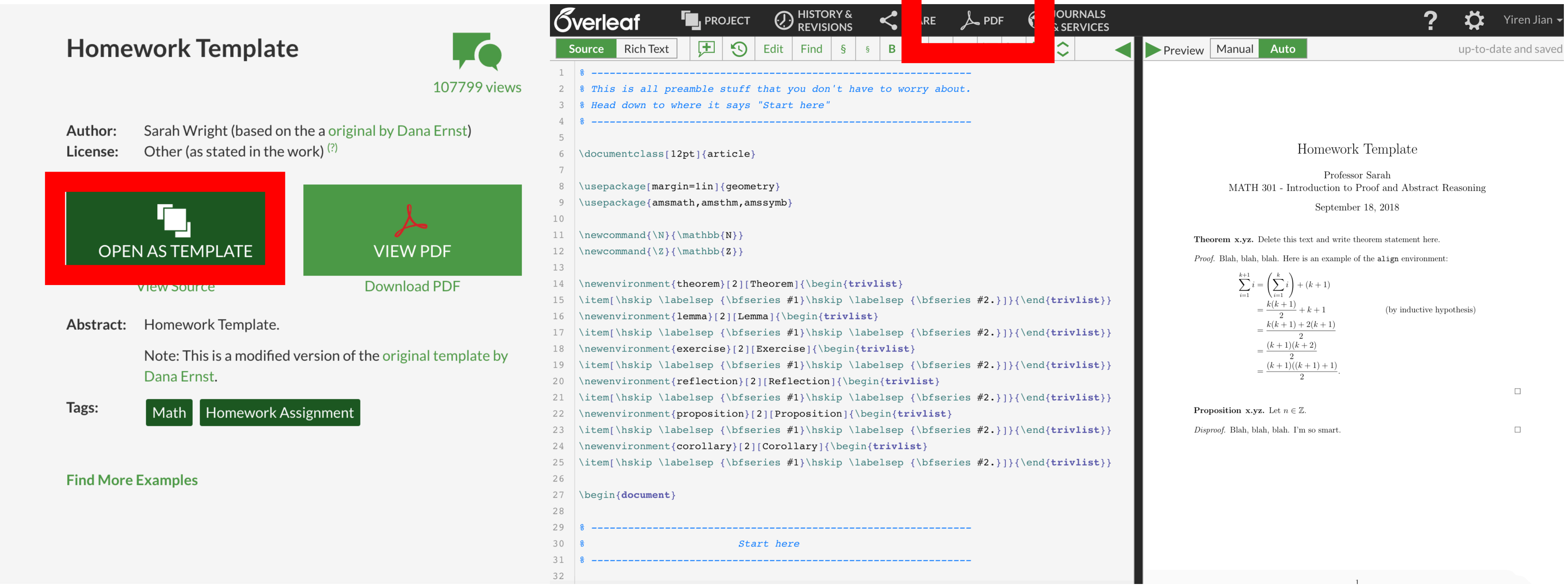


Start your projects with quality LaTeX templates for journals, CVs, resumes, papers, project reports, and more. Search or browse below.

Search...

After you register your account, choose a template from homework assignment.

You can download the PDF file once you finish.



Homework Template

107799 views

Author: Sarah Wright (based on the a original by Dana Ernst)
License: Other (as stated in the work) (?)

Abstract: Homework Template.

Note: This is a modified version of the original template by Dana Ernst.

Tags: Math | Homework Assignment

```
% ----------------------------------------------------------------
% This is all preamble stuff that you don't have to worry about.
% Head down to where it says "Start here"
% ----------------------------------------------------------------

\documentclass[12pt]{article}

\usepackage[margin=1in]{geometry}
\usepackage{amsmath,amsthm,amssymb}

\newcommand{\N}{\mathbb{N}}
\newcommand{\Z}{\mathbb{Z}}

\newenvironment{theorem}[2][Theorem]{\begin{trivlist}
\item[\hskip \labelsep {\bfseries #1}\hskip \labelsep {\bfseries #2.}]}{\end{trivlist}}
\newenvironment{lemma}[2][Lemma]{\begin{trivlist}
\item[\hskip \labelsep {\bfseries #1}\hskip \labelsep {\bfseries #2.}]}{\end{trivlist}}
\newenvironment{exercise}[2][Exercise]{\begin{trivlist}
\item[\hskip \labelsep {\bfseries #1}\hskip \labelsep {\bfseries #2.}]}{\end{trivlist}}
\newenvironment{reflection}[2][Reflection]{\begin{trivlist}
\item[\hskip \labelsep {\bfseries #1}\hskip \labelsep {\bfseries #2.}]}{\end{trivlist}}
\newenvironment{proposition}[2][Proposition]{\begin{trivlist}
\item[\hskip \labelsep {\bfseries #1}\hskip \labelsep {\bfseries #2.}]}{\end{trivlist}}
\newenvironment{corollary}[2][Corollary]{\begin{trivlist}
\item[\hskip \labelsep {\bfseries #1}\hskip \labelsep {\bfseries #2.}]}{\end{trivlist}}

\begin{document}

% ----------------------------------------------------------------
%                         Start here
% ----------------------------------------------------------------

```

Homework Template

Professor Sarah
MATH 301 - Introduction to Proof and Abstract Reasoning

September 18, 2018

Theorem x.yz. Delete this text and write theorem statement here.

Proof. Blah, blah, blah. Here is an example of the `align` environment:

$$\begin{aligned}\sum_{i=1}^{k+1} i &= \left(\sum_{i=1}^{k} i\right) + (k+1) \\ &= \frac{k(k+1)}{2} + k + 1 && \text{(by inductive hypothesis)} \\ &= \frac{k(k+1)+2(k+1)}{2} \\ &= \frac{(k+1)(k+2)}{2} \\ &= \frac{(k+1)((k+1)+1)}{2}.\end{aligned}$$

□

Proposition x.yz. Let $n \in \mathbb{Z}$.

Disproof. Blah, blah, blah. I'm so smart. □

You edit your text here and it will automatically render the PDF file for you.

http://physics.clarku.edu/sip/tutorials/TeX/intro.html

This is a good example showing how to make equation, tables, lists, figures and special symbols. And you can always ask help from Google.