AWS MACHINE LEARNING FOUNDATIONS COURSE

Contents

Software Practices I

Writing clean code

Use meaningful names, and proper whitespace.

Writing modular code

- DRY (Don't repeat yourself)
- Abstract out logic to improve readability—like into a function
- Minimize number of entities
- Functions should do one thing—Single responsibility principle
- Arbitrary variable names can be more effective in certain functions
- Minimize number of arguments to a functions to 3

Writing efficient code

- Code that runs infrequently for a short time need not be highly optimized
- Code that needs to run fast, such as a live feed, should be highly optimized
- Code can be refactored to be optimized after an initial solution
- Use vector operations over loops whenever possible
- Refactor using different data structures to make code more efficient
- When searching for solutions, it's better to experiment with different solutions to find methods that are optimum, rather than stick with the most popular solution

Related links: What makes sets faster than lists?

optimizing_code_common_books

1. Using NumPy and it's intersect1d instead of lists and loops makes a difference of 1386.01 times speed increase
2. Using sets over lists and their intersection method makes a difference of 4595.53 times speed increase
3. Set and intersection() is 3.315 times faster than using NumPy and it's intersect1d

Related links: numpy.intersect1d and Intersection() function Python - GeeksforGeeks

optimizing_code_holiday_gifts

1. Arithmetic operations can be optimized over numpy arrays. Scalar values (entire rows) or vectors can be easily all multiplies, divied, added to, subtracted from, etc. much faster than using loops or other iterations

Related links: 1.4.2. Numerical operations on arrays, How do I select elements of an array given condition?, and numpy.sum

Documentation

1. Using inline comments to add line level docs
    a. Useful for explaining code when code can't speak
2. Using doc strings to add docs at the function and module level
    a. They can be one line to explain a single function
    b. Multiline docstrings have more parts such as Arguments: , Returns: , and a longer descriptions

Version Control

Version control can be used to store, retrieve, and search through changes in a project. It helps protect the project developers from losing work, and takes care of the work of managing versions and change control.

Software Practices II

Testing

1. Unit tests are used to test small units of code
2. Pytest is used to process tests
3. We should only have one assert statement per test function
4. Pytest stops if there are syntax errors

Test Driven Development

1. Writing tests before writing implementation code
2. Tests can check for all the different scenarios and edge cases you can think of, before even starting to write your function
3. You can also write better tests this way as your program evolves, rather than writing one hurried test at the end
4. When refactoring or adding to your code, tests help you rest assured that the rest of your code didn't break while you were making those changes (regression testing).

Logging

Logging is valuable for understanding the events that occur while running your program. For example, if you run your model over night and see that it's producing ridiculous results the next day, log messages can really help you understand more about the context in which this occurred.

Code Review

When your coworker finishes up some code that they want to merge to the team's code base, they might send it to you for review. You provide feedback and suggestions, and then they may make changes and send it back to you. When you are happy with the code, you approve and it gets merged to the team's code base.

Explain issues and make suggestions

BAD: Make model evaluation code its own module - too repetitive.

BETTER: Make the model evaluation code its own module. This will simplify models.py to be less repetitive and focus primarily on building models.

GOOD: How about we consider making the model evaluation code its own module? This would simplify models.py to only include code for building models. Organizing these evaluations methods into separate functions would also allow us to reuse them with different models without repeating code.

Keep comments objective

BAD: I wouldn't groupby genre twice like you did here... Just compute it once and use that for your aggregations.

BAD: You create this groupby dataframe twice here. Just compute it once, save it as groupby_genre and then use that to get your average prices and views.

GOOD: Can we group by genre at the beginning of the function and then save that as a groupby object? We could then reference that object to get the average prices and views without computing groupby twice.

It helps if you remember that it's a group effort, and that it would be better to frame it as something we all would like to get done, since it's something we would have to work on later as well.

Provide code examples

Let's say you were reviewing code that included the following lines:

```
first_names = []
```

```
last_names = []

for name in enumerate(df.name):
    first, last = name.split(' ')
    first_names.append(first)
    last_names.append(last)

df['first_name'] = first_names
df['last_names'] = last_names
```

BAD: You can do this all in one step by using the pandas str.split method.

GOOD: We can actually simplify this step to the line below using the pandas str.split method. Found this on this stack overflow post: https://stackoverflow.com/questions/14745022/how-to-split-a-column-into-two-columns

```
df['first_name'], df['last_name'] = df['name'].str.split(' ', 1).str
```

Object Oriented Programming

Introduction

Objects have attributes and methods, or characteristics and actions.

A generic version of an object is it's class. Classes are blueprints for creating objects.

'self ' is a self-referential pointer, much like 'this'. It is used to refer to the memory location of the object itself. Using '.' notation, we can access the attributes and methods of a particular object.

'__init__' is used to initialize the values of the object upon object creation.

It is easier to modularize code using classes, where classes are stored in separate files, and then called to create objects in other files.

Classes and Objects

shirt_exercise

1. Create 2 objects 'shirt_one' and 'shirt_two' with given values
2. Modify shirt one with 'change_price' and find it's 12% discount price with 'discount'
3. Find the total price of the two objects and store in 'total'
4. Find the 14% and 6% discount prices of these 2 objects and store in 'total_discount'
5. Run tests using the cell at the end of the doc

Get and Set methods

A get method is for obtaining an attribute value. A set method is for changing an attribute value.

Example:

```
class Shirt:

    def __init__(self, shirt_color, shirt_size, shirt_style, shirt_price):
        self._price = shirt_price

    def get_price(self):
      return self._price

    def set_price(self, new_price):
      self._price = new_price
shirt_one = Shirt('yellow', 'M', 'long-sleeve', 15)
print(shirt_one.get_price())
shirt_one.set_price(10)
```

One of the benefits of set and get methods is that, as previously mentioned in the course, you can hide the implementation from your user. Maybe originally a variable was coded as a list and later became a dictionary. With set and get methods, you could easily change how that variable gets accessed. Without set and get methods, you'd have to go to every place in the code that accessed the variable directly and change the code.

OOP Syntax Exercise - Part 2

1. Every method in the class must have 'self' as the first argument

Python Package: Gaussian Distribution

Python Package that will be able to

1. Read dataset
2. Calculate mean, standard deviation
3. Plot histogram and probability density function
4. Add 2 Gaussian Distributions

Later, it will be extended to work with binomial distributions

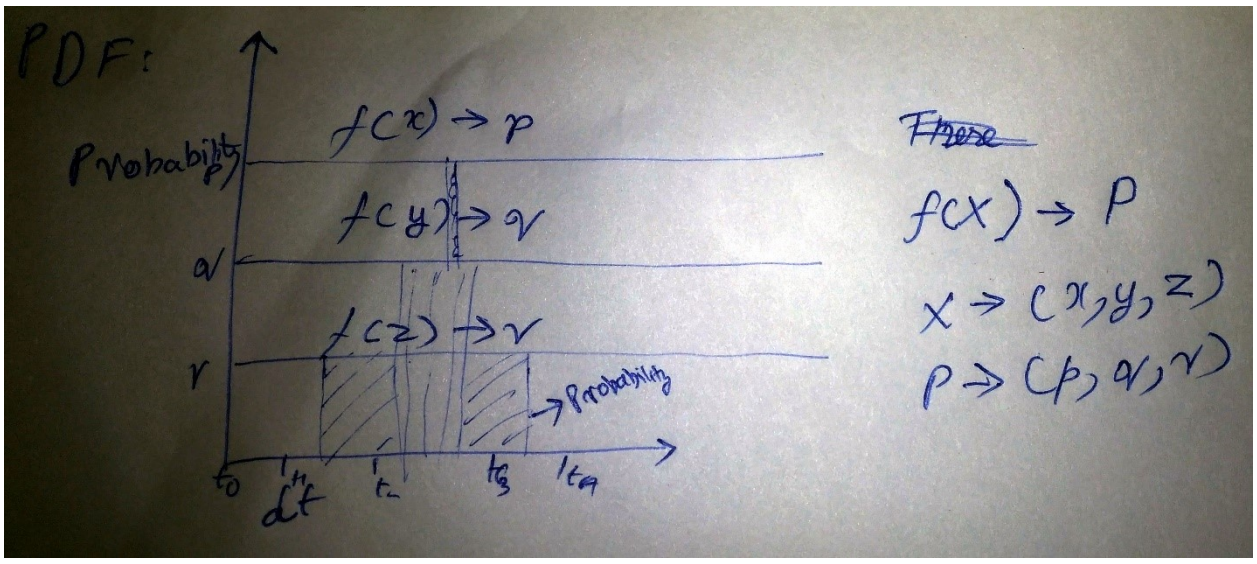
Related links: Probability density function, Normal distribution, Binomial distribution, Variance, Standard deviation, How to do Normal Distributions Calculations, 14. Normal Probability Distributions

(Wikipedia, Probability density function) (Wikipedia, Normal distribution), (Wikipedia, Binomial distribution) (Wikipedia, Variance), (Wikipedia, Standard deviation) (Mathematics), (Laerd)

Theory

Probability Density Function:

n probability theory, a probability density function (PDF), or density of a continuous random variable, is a function whose value at any given sample (or point) in the sample space (the set of possible values taken by the random variable) can be interpreted as providing a relative likelihood that the value of the random variable would equal that sample.[2] In other words, while the absolute likelihood for a continuous random variable to take on any particular value is 0 (since there are an infinite set of possible values to begin with), the value of the PDF at two different samples can be used to infer, in any particular draw of the random variable, how much more likely it is that the random variable would equal one sample compared to the other sample.



In a more precise sense, the PDF is used to specify the probability of the random variable falling within a particular range of values, as opposed to taking on any one value. This probability is given by the integral of this variable's PDF over that range—that is, it is given by the area under the density function but above the horizontal axis and between the lowest and greatest values of the range. The probability density function is nonnegative everywhere, and its integral over the entire space is equal to 1.

Example

Suppose bacteria of a certain species typically live 4 to 6 hours. The probability that a bacterium lives exactly 5 hours is equal to zero. A lot of bacteria live for approximately 5 hours, but there is no chance that any given bacterium dies at exactly 5.00000000000... hours. However, the probability that the bacterium dies between 5 hours and 5.01 hours is quantifiable. Suppose the answer is 0.02 (i.e., 2%). Then, the probability that the bacterium dies between 5 hours and 5.001 hours should be about 0.002, since this time interval is one-tenth as long as the previous. The probability that the bacterium dies between 5 hours and 5.0001 hours should be about 0.0002, and so on.

In these three examples, the ratio (probability of dying during an interval) / (duration of the interval) is approximately constant, and equal to 2 per hour (or 2 hour−1). For example, there is 0.02 probability of dying in the 0.01-hour interval between 5 and 5.01 hours, and (0.02 probability / 0.01 hours) = 2 hour−1. This quantity 2 hour−1 is called the probability density for dying at around 5 hours. Therefore, the probability that the bacterium dies at 5 hours can be written as (2 hour−1) dt. This is the probability that the bacterium dies within an infinitesimal window of time around 5 hours, where dt is the duration of this window. For example, the probability that it lives longer than 5 hours, but shorter than (5 hours + 1 nanosecond), is (2 hour−1)×(1 nanosecond) ≈ 6×10−13 (using the unit conversion 3.6×1012 nanoseconds = 1 hour).

There is a probability density function f with f(5 hours) = 2 hour−1. The integral of f over any window of time (not only infinitesimal windows but also large windows) is the probability that the bacterium dies in that window.

Variance

In probability theory and statistics, variance is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of numbers are spread out from their average value.

The variance of a random variable X is the expected value of the squared deviation from the mean of X, $\mu = \mathrm{E}[X]$:

$$\operatorname{Var}(X) = \mathrm{E}\left[(X - \mu)^2\right].$$

If the generator of random variable X is discrete with probability mass function $x_1 \mapsto p_1, x_2 \mapsto p_2, \ldots, x_n \mapsto p_n$ then

$$\operatorname{Var}(X) = \sum_{i=1}^{n} p_i \cdot (x_i - \mu)^2,$$

If the random variable X has a probability density function $f(x)$, and $F(x)$ is the corresponding cumulative distribution function, then

$$\operatorname{Var}(X) = \sigma^2 = \int_{\mathbb{R}} (x - \mu)^2 f(x)\, dx$$

Standard deviation

In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values.[1] A low standard deviation indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the values are spread out over a wider range.

The standard deviation of a random variable, statistical population, data set, or probability distribution is the square root of its variance. It is algebraically simpler, though in practice less robust, than the average absolute deviation.[2][3] A useful property of the standard deviation is that, unlike the variance, it is expressed in the same units as the data.

In addition to expressing the variability of a population, the standard deviation is commonly used to measure confidence in statistical conclusions. For example, the margin of error in polling data is determined by calculating the expected standard deviation in the results if the same poll were to be conducted multiple times. This derivation of a standard deviation is often called the "standard error" of the estimate or "standard error of the mean" when referring to a mean. It is computed as the standard deviation of all the means that would be computed from that population if an infinite number of samples were drawn and a mean for each sample were computed.

Let X be a random variable with mean value μ:

$$\mathrm{E}[X] = \mu.$$

Here the operator E denotes the average or expected value of X. Then the standard deviation of X is the quantity

$$\sigma = \sqrt{\mathrm{E}\left[(X - \mu)^2\right]}$$

In the case where X takes random values from a finite data set $x_1, x_2, \ldots, x_N$, with each value having the same probability, the standard deviation is

$$\sigma = \sqrt{\sum_{i=1}^{N} p_i (x_i - \mu)^2}, \text{ where } \mu = \sum_{i=1}^{N} p_i x_i .$$

The standard deviation of a continuous real-valued random variable X with probability density function p(x) is

$$\sigma = \sqrt{\int_{\mathbf{X}} (x - \mu)^2 \, p(x) \, \mathrm{d}x}, \text{ where } \mu = \int_{\mathbf{X}} x \, p(x) \, \mathrm{d}x,$$

Normal distribution

In probability theory, a normal (or Gaussian or Gauss or Laplace–Gauss) distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The parameter μ is the mean or expectation of the distribution (and also its median and mode); and σ is its standard deviation. The variance of the distribution is σ^2. A random variable with a Gaussian distribution is said to be **normally distributed** and is called a **normal deviate**.

The simplest case of a normal distribution is known as the standard normal distribution. This is a special case μ=0 and σ=1, and it is described by this probability density function:

$$\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$

Standard deviation of one means there is a width of 1 between each band in the curve.

The factor $1/\sqrt{2\pi}$ in this expression ensures that the total area under the curve $\varphi(x)$ is equal to one.[note 1] The factor $1/2$ in the exponent ensures that the distribution has unit variance (i.e., the variance is equal to one), and therefore also unit standard deviation. This function is symmetric around $x = 0$, where it attains its maximum value $1/\sqrt{2\pi}$ and has inflection points at $x = +1$ and $x = -1$.

This means that the PDF generates, for each x, a probability p, and the plot of those p is the gaussian distribution.

To calculate the probability:

It was found that the mean length of 100 parts produced by a lathe was 20.05 mm with a standard deviation of 0.02 mm. Find the probability that a part selected at random would have a length

(a) between 20.03 mm and 20.08 mm

(a) 20.03 is 1 standard deviation below the mean;

20.08 is $\dfrac{20.08 - 20.05}{0.02} = 1.5$ standard deviations above the mean.

$P(20.03 < X < 20.08)$

$= P(-1 < Z < 1.5)$

$= 0.3413 + 0.4332$

$= 0.7745$

So the probability is 0.7745.

Binomial distribution

In probability theory and statistics, the binomial distribution with parameters n and p is the discrete probability distribution of the number of successes in a sequence of n independent experiments, each asking a yes–no question, and each with its own boolean-valued outcome: success/yes/true/one (with probability p) or failure/no/false/zero (with probability q = 1 − p). A single success/failure experiment is also called a Bernoulli trial or Bernoulli experiment and a sequence of outcomes is called a Bernoulli process; for a single trial, i.e., n = 1, the binomial distribution is a Bernoulli distribution. The binomial distribution is the basis for the popular binomial test of statistical significance.

The binomial distribution is frequently used to model the number of successes in a sample of size n drawn with replacement from a population of size N. If the sampling is carried out without replacement, the draws are not independent and so the resulting distribution is a hypergeometric distribution, not a binomial one. However, for N much larger than n, the binomial distribution remains a good approximation, and is widely used.

In general, if the random variable X follows the binomial distribution with parameters n ∈ ℕ and p ∈ [0,1], we write X ~ B(n, p). The probability of getting exactly k successes in n independent Bernoulli trials is given by the probability mass function:

$$f(k,n,p) = \Pr(k;n,p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

for k = 0, 1, 2, ..., n, where

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

is the binomial coefficient, hence the name of the distribution. The formula can be understood as follows. k successes occur with probability pk and n – k failures occur with probability (1 – p)n – k. However, the k successes can occur anywhere among the n trials, and there ${}^{n}C_{k}$ different ways of distributing k successes in a sequence of n trials.

This means that for each PDF value of x, it generates k successes with probability p. Plotting those probabilities gives us the Binomial Distribution plot.

The Gaussian Class

The class stores:

1. Mean
2. Standard Deviation
3. The dataset

It has functions for:

1. Loading data
2. Calculating mean
3. Calculating standard dev
4. Plotting a histogram
5. Calculating the probability density function

gaussian_code_exercise.ipynb

1. SQRT function is in math.sqrt
2. Mean of a data set is the simple average, calculated by summing the row and then dividing it by the number of records
3. Standard deviation differs between a population and a sample of a population
4. For the standard deviation of the population
    a. Calculate the mean
    b. Calculate the squared difference by subtracting the mean from each value and then squaring the difference
    c. Calculate the mean of the squared difference by simple average of the values
    d. Calculate the standard deviation by finding the square root of the mean squared difference
    e. $$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$
5. For the standard deviation of a sample of the population
    a. Calculate the mean
    b. Calculate the squared difference by subtracting the mean from each value and then squaring the difference
    c. Calculate the mean of the squared difference by dividing the N values of the squared differences by N-1
    d. Calculate the standard deviation by finding the square root of the mean squared difference
    e. $$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$
6. Create a histogram of data using plt.hist()

Observations from answers.py

1. Self.stdev() can be refactored to be less repetitive
2. I did better on readability on calculating the PDF function
3. I did better in making sure I was getting only floating-point values

Magic Methods in code

The aim is to be able to use overriding of default python behavior to implement addition of 2 objects of the Gaussian class.

Magic methods allow us to customize default python behavior. For example, the __init__ method can customize how python instantiates an object. __add__ can be used to add 2 objects, and __repr__ can be used to change how the object is represented when printed.

magic_methods

1. Add the means to get mean value, assign that to a new object's mean and return it
2. Sqrt the sum of squared values of standard deviations, assign that to a new object's standard deviation and return it
3. Return a string as part of __repr__

Inheritance

Inheriance is where a choild class inherits methods and attributes from it's parent class.

Updates to the parent class trickle down to the child class.

inheritance_exercise_clothing

1. Create a new class Blouse
2. Add methods to new class: triple_price
3. Add methods to parent class—this is called in the child class as well: calculate_shipping
4. Make sure to call constructor of parent class in child class

Inheritance Gaussian Class

The distribution super class is used as the base class for the Gaussian and the Binomial classes.

inheritance_probability_distribution

1. The distribution class is the parent class for the gaussian class
2. The code has been refactored to adjust the gaussian class init method—it now initializes the distribution class
3. The read data method has been overridden in the gaussian class
4. The functionality remains the same throughout

Advanced OOP topics

Related links: Python's Instance, Class, and Static Methods Demystified, Class and Instance Attributes, Mixins for Fun and Profit, Primer on Python Decorators

(Python, Python's Instance, Class, and Static Methods Demystified), (Python, Primer on Python Decorators), (Class and Instance Attributes), (Mixins for Fun and Profit)

Static, Class, and Instance methods

- Classes are objects themselves
- Instance methods are able to modify instance state, as well as class state.
- Class methods can modify class state only, but not the state of any instance
- Class methods can be used to create alternate constructors for a class
- Class methods are used in factory methods
- Static methods can't modify class or instance state, and act like regular functions
- Static methods are a way for namespacing methods, they belong to the class's namespace
- Static and class methods communicate and (to a certain degree) enforce developer intent about class design. This can have maintenance benefits.

Class, Instance, and Static method example

```
class MyClass:
    def method(self):
        return 'instance method called', self

    @classmethod
    def classmethod(cls):
        return 'class method called', cls

    @staticmethod
    def staticmethod():
        return 'static method called'
```

Class attributes and instance attributes

Instance attributes are owned by the specific instances of a class. That is, for two different instances, the instance attributes are usually different. Class attributes stay with class objects, while instance attributes are unique to all instances. Class attributes are attributes which are owned by the class itself. They will be shared by all the instances of the class. Therefore they have the same value for every instance. We define class attributes outside all the methods, usually they are placed at the top, right below the class header.

```
class A:
    a = "I am a class attribute!"

x = A()
y = A()
x.a
```

Output:: `'I am a class attribute!'`

```
A.a
```

Output:: `'I am a class attribute!'`

FIGURE 1 CLASS ATTRIBUTE EXAMPLE

Class attributes are stored in the `.__class__.__dict__` dictionary while the instance attributes are stored in the `.__dict__` dictionary.

Class attribute and method example

```
class Pet:
    _class_info = "pet animals"

    @classmethod
    def about(cls):
        print("This class is about " + cls._class_info + "!")


class Dog(Pet):
    _class_info = "man's best friends"

class Cat(Pet):
    _class_info = "all kinds of cats"

Pet.about()
Dog.about()
Cat.about()
```

```
This class is about pet animals!
This class is about man's best friends!
This class is about all kinds of cats!
```

Mixins

Mixins are classes that encapsulate a specific functionality. They can be inherited to provide specific behavour to an object—not exactly a parent class, more like an enhanced import of functionality—"mixing in functionality".

Mixin examples

Example 1

```
import logging


class LoggerMixin(object):
    @property
    def logger(self):
        name = '.'.join([
            self.__module__,
            self.__class__.__name__
        ])
        return logging.getLogger(name)
```

```
class EssentialFunctioner(LoggerMixin, object):
    def do_the_thing(self):
        try:
            ...
        except BadThing:
            self.logger.error('OH NOES')


class BusinessLogicer(LoggerMixin, object):
    def __init__(self):
        super().__init__()
        self.logger.debug('Giving the logic the business...')
```

Example 2

```
from django.utils.decorators import decorator_from_middleware,
method_decorator
from django.views import generic


network_protected = decorator_from_middleware(
    middleware.NetworkProtectionMiddleware
)


class NetworkProtectionMixin(object):
    @method_decorator(network_protected)
    def dispatch(self, *args, **kwargs):
        return super().dispatch(*args, **kwargs)


class SecretView(NetworkProtectionMixin, generic.View):
    ...
```

Python Decorators

Placeholder for notes about python decorators.

Modularized Code

1. The Distribution class has been shifted to the GeneralDistribution.py file.
2. The GaussianDistribution.py file contains the Gaussian class
3. example_code.py is used to create a Gaussian object and print it's mean

Packages

Packages have a fixed structure. They need an __init__.py file to initialize, and a setup.py file to install.

Multiple modules make up a package. The name of the package should be the same as the name of the folder the modules are in, along with the __init__.py file.

setup.py has metadata about the package, such as name, version number, description, etc. To install a local python package, type **pip install .** in the folder where the package is stored.

To see where the package has been installed, type **package-name**.__file__ .

Virtual Environments

Conda can be used to create environments.

```
python3 -m venv environmentname
source environmentname/bin/activate
pip install numpy
```

pip and venv will be used for this exercise.

Exercise: Making a Package and Pip Installing

1. Copy files into directory 3a_python_package with **cp -i example_code.py Gaussiandistribution.py Generaldistribution.py /home/workspace/3a_python_package**
2. Create a new folder with **mkdir 'distributions'**
3. Copy files to distributions folder with **cp -i example_code.py Gaussiandistribution.py Generaldistribution.py distributions**
4. Remove files with **rm example_code.py Gaussiandistribution.py Generaldistribution.py** from 3a_python_package
5. Create __init__.py file and add **from .Gaussiandistribution import Gaussian** to it in **'distributions'** folder
6. Create setup.py file in folder
7. Add a . and change the import to **from .Generaldistribution import Distribution** in **Gaussiandistribution.py**
8. Add the following code to setup.py

   ```
   from setuptools import setup


   setup(name = 'distributions',
       version = '0.1',
       description = 'Gaussian Distribution',
       packages = ['distributions'],
       zip_safe = False)
   ```

9. Install package with **pip install .**
10. Try using the package in the python interpreter

Binomial Class

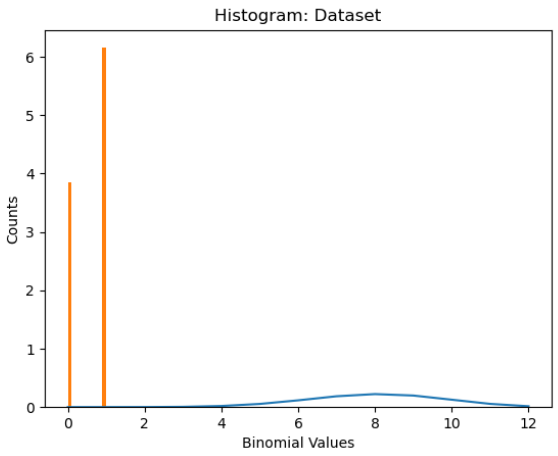
I need to create a class that can perform the same functions as in the Gaussian class, but with binomial distributions.

Exercise: Binomial Class

There are 2 files, BinomialDistribution.py and BinomialDistribution_challenge.py.

The first one contains some pre-written code, while the other one doesn't. I intend to use the first one to give structure to the second one. Then, I'll work on the code.

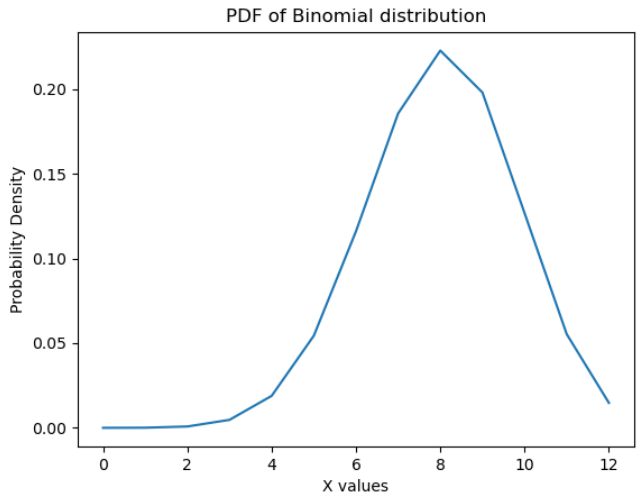
1. Mean is calculated as the multiplication of n and p. For standard deviation, the formula sqrt(n * p * (1 - p)) is used.
2. Calculate n and p in the dataset by using count() on the list to find p and len() to find n.
3. Plot the histogram of the dataset using density as the function.



4.

$$f(k,n,p) = \Pr(k;n,p) = \Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

5. Calculate the PDF a using the formula .
6. Plot the graph of the PDF values for k from 1..n.



7.
8. Write a __add__ magic method to add 2 binomial distributions, by adding their n values, checking for their datasets and concatenating the data.
9. Write a __repr__ magic method to return stats about the distribution.

PyPi

It's a python package repository used to store packages on the internet. TestPyPi is used for storing test packages before release.

The exercise has been skipped: Exercise: Upload to PyPi

Machine Learning with AWS DeepComposer

This will focus on ML with AWS, and will involve an overview, followed by using AWS DeepComposer, and a GAN.

There are various ML and AI services, that can enable ML development on AWS seamlessly, such as SageMaker, DeepComposer, DeepLens, etc.

ML Techniques and Generative AI

Generative AI is the technique of creating new objects based on sample inputs. The adversarial technique uses one AI to create stuff, and the other to inspect it for accuracy.

Used to deisgn aircraft parts, new spacecraft, and teeth. This is mostly an unsupervised technique.

AWS Composer and Generative AI

AWS Deep Composer uses Generative AI, or specifically Generative Adversarial Networks (GANs), to generate music. GANs pit 2 networks, a generator and a discriminator, against each other to generate new content.

It is a 32 key 2 octave key device, and there is a replica of the keyboard right inside the AWS management console. There is a place to upload an input melody, and a model selector.

The resulting melody can be stored as MIDI files.

1. To get to the main AWS DeepComposer console, navigate to AWS DeepComposer. Make sure you are in the US East-1 region.
2. Once there, click on **Get started**
3. In the left hand menu, select **Music studio** to navigate to the DeepComposer music studio
4. To generate music you can use a virtual keyboard or the physical AWS DeepComposer keyboard. For this lab, we'll use the virtual keyboard.
5. Choose GAN as the generative AI technique
6. To view sample melody options, select the drop down arrow next to **Input**
7. Select **Twinkle, Twinkle, Little Star**
8. Next, choose a model to apply to the melody by clicking **Select model**
9. From the sample models, choose **Rock** and then click **Select model**
10. Next, select **Generate composition**. The model will take the 1 track melody and create a multitrack composition (in this case, it created 4 tracks)
11. Click **play** to hear the output

Generative AI algorithms

Linked Resources: Basics of generative AI and GANs, Introduction to generative adversarial networks (GANs), Introduction to autoregressive convolutional neural networks (AR-CNNs).

(Amazon, Basics of generative AI and GANs) , (Amazon, Introduction to autoregressive convolutional neural networks (AR-CNNs)), (Amazon, Introduction to generative adversarial networks (GANs)).

Generative AI is a broad category of algorithms, the most popular of which include Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs) and Autoregressive (AR) Models.

Generative adversarial networks

Generative Adversarial Networks (GANs) are a type of machine learning network in which two neural networks compete. One network is tasked with generating realistic-seeming content (unsupervised learning), while the other network is tasked with distinguishing the generated content against real data (supervised learning).

The generator

The generator within a GAN is a machine learning model that's trained to produce realistic-seeming output. The Generator is like an orchestra — it trains, practices, and tries to generate polished music.

The discriminator

The discriminator is another machine learning model which is trained to take an input and classify whether or not the input is real or generated. The discriminator is like the orchestra's conductor — it judges the quality of the output and tries to achieve the style of music that it has been trained on.

Content generation

Initially, the generator produces content samples based on random inputs.

Classifying content

The discriminator looks for features (e.g. tempo) from the dataset it was trained on (e.g. Pop, Rock, Classical) in the content samples the generator created. It decides whether or not the content samples belong in the training set.

Training GANs

The results of the discriminator's judgements are used to train both models. The generator is trained to optimize for producing realistic content that the discriminator cannot distinguish from the real samples. Meanwhile, the discriminator is trained to increase its ability to detect generated content. This back-and-forth behavior, where the two models are directly competing against each other, is the adversarial part of GANs.

Introduction to computer music

For a machine learning model to be able to interpret and generate music, it's important to represent music in a format which preserves the precise details of how the music is played. Instead of processing a sound file, machine learning models such as those used in AWS DeepComposer examine those details to faithfully reproduce instruments and musical styles.

Pitch

Pitch is a tone that is assigned a relative position on a musical scale. Each note is assigned a numeric value, starting at 0 for the lowest possible note, and ranging up to 127 for the highest. The keys on the AWS DeepComposer keyboard range from 41 to 72. The Octave Adjust buttons shift the pitch values up or down in multiples of 12 to play higher or lower pitches.

Velocity

Velocity encodes how hard a single note is pressed. Pressing the key faster results in a higher value for the velocity, which creates a louder sound. Velocity values range from 1 (minimum, practically inaudible) to 127 (maximum).

Tempo

Tempo describes how fast music is played. Music typically follows a certain beat or meter, which drives the rhythm of the notes played. The speed of this beat is measured in beats per minute. A higher number of beats per minute corresponds to a faster playback speed (tempo).

MIDI

The MIDI file format is an industry-standard used by computers to record and store music. The file format encodes details such as playback tempo, which instruments are used, and a series of events that encode a note being pressed or released, including the pitch and velocity of the note.

Piano-roll visualization

The MIDI format can be partially represented using piano-roll visualization, which can help you visualize the data. Time is represented in the horizontal dimension and pitch is represented in the vertical dimension. Bars represent notes being pressed. The start of the bar is the start of the note being played, and the end of the bar is the note being released

Exercise: Generate an Interface

1. Follow given instructions to generate a model

How DeepComposer Works

Uses a GAN to generate accompaniments.

Loss functions are used to monitor the generator and the discriminator, which monitors how close the model comes to a desired output. In GANs there is early fluctuation, and then it stabilizes to a point called convergence. The models are trained for many cycles called epochs.
The weights are used to give importance to attributes, and the activation function results in a sum of these weighted attributes, which gives the output.
The deviation of the output from the desired or true result is called the loss. Ideally, as the weights update, the model improves making less and less errors.
Convergence happens once the loss functions stabilize.

How DeepComposer works

1. Input melody captured on the AWS DeepComposer console
2. Console makes a backend call to AWS DeepComposer APIs that triggers an execution Lambda.
3. Book-keeping is recorded in Dynamo DB.
4. The execution Lambda performs an inference query to SageMaker which hosts the model and the training inference container.
5. The query is run on the Generative AI model.
6. The model generates a composition.
7. The generated composition is returned.
8. The user can hear the composition in the console.

Training Architecture

Aside from convergence, we use similarity index to measure model performance, which measures how close our model's data comes to the original.

1. User launch a training job from the AWS DeepComposer console by selecting hyperparameters and data set filtering tags
2. The backend consists of an API Layer (API gateway and lambda) write request to DynamoDB
3. Triggers a lambda function that starts the training workflow
4. It then uses AWS Step Funcitons to launch the training job on Amazon SageMaker
5. Status is continually monitored and updated to DynamoDB
6. The console continues to poll the backend for the status of the training job and update the results live so users can see how the model is learning

Challenges with GANs

- Clean datasets are hard to obtain
- Not all melodies sound good in all genres
- Convergence in GAN is tricky – it can be fleeting rather than being a stable state
- Complexity in defining meaningful quantitive metrics to measure the quality of music created

Generative AI

A generator is a convolutional neural network (CNN) that learns to create new data resembling the source data it was trained on. The discriminator is another convolutional neural network (CNN) that is trained to differentiate between real and synthetic data. Discriminator and Generators work asynchronously, trained in alternating cycles such that the generator learns to produce more and more realistic data while the discriminator iteratively gets better at learning to differentiate real data from the newly created data.

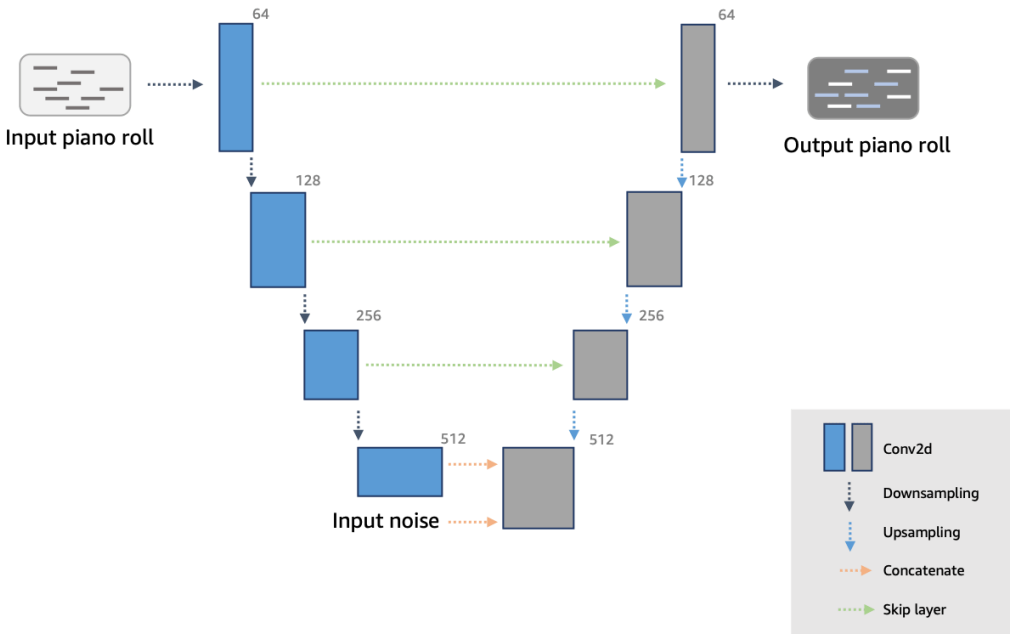
Introduction to U-Net Architecture

he U-Net architecture uses a publicly available dataset of Bach's compositions for training the GAN. In AWS DeepComposer, the generator network learns to produce realistic Bach-syle music while the discriminator uses real Bach music to differentiate between real music compositions and newly created ones

Model Architecture

Discriminators are also referred to as critics because they evaluate the input from the generator and provide feedback, the discriminator's role is solely focused on determining if the input is realistic. The discriminator returns a scalar output value, of how real the output is. It is composed of four convolutional layers and a dense layer at the end.

The generator is adapted from U-net architecture, and consists of an encoder that encodes the input to a low level latent space, which the decoder translates to multi=track music output. Along with the input a noise vector is also provided as input, to ensure that every output is unique. The architecture forms a U shape ans thus it is called U-net.



Training Methodology

The gneenrator and discriminator work asynchronously in a tight loop.

The generator takes single track input, and produces multi-track outputs by adding varied accompaniments to the input. The discriminator predicts how real these tracks are compared to tracks in the training dataset.

The generator updates it's weights according to scalar output value from the discriminator. As the generator gets better at creating music accompaniments, it begins fooling the discriminator. So, the discriminator needs to be retrained as well.

This continues for multiple epochs. When the training loop has passed through the entire training dataset once, we call that one epoch. Training for a higher number of epochs will mean your model will take longer to complete its training task, but it may produce better output if it has not yet converged.

Learning Rate

The learning rate controls how rapidly the weights and biases of each network are updated during training. A higher learning rate might allow the network to explore a wider set of model weights, but might pass over more optimal weights.

Update ratio

A ratio of the number of times the discriminator is updated per generator training epoch. Updating the discriminator multiple times per generator training epoch is useful because it can improve the discriminators accuracy. Changing this ratio might allow the generator to learn more quickly early-on, but will increase the overall training time.

Evaluation

In GANs, the discriminator loss has been found to correlate well with sample quality. The discriminator loss should converge to o, while the generator loss should to converge to some number which need not be zero. The model training stops when the loss function plateaus.

Loss is useful as an evaluation metric since the model will not improve as much or stop improving entirely when the loss plateaus. At that point sample output might feature elements of the training dataset.

It is still difficult to measure accuracy for GANs, so domain specific measurements are used.

Inference

After training the generator model can be run to generate new accompaniments for a given input melody.

The final process for music generation then is as follows:

1. Transform single-track music input into piano roll format.
2. Create a series of random numbers to represent the random noise vector.
3. Pass these as input to our trained generator model, producing a series of output piano rolls. Each output piano roll represents some instrument in the composition.
4. Transform the series of piano rolls back into a common music format (MIDI), assigning an instrument for each track.

Build a Custom GAN

Setup the SageMaker instance.

The data is in the format of a MIDI file, so convert it into 2D images. Our data is stored as a NumPy Array, or grid of values. Our dataset comprises 229 samples of 4 tracks (all tracks are piano). Each sample is a 32 time-step snippet of a song, so our dataset has a shape of:

(num_samples, time_steps, pitch_range, tracks)

or

(229, 32, 128, 4)

The discriminator is trained on this dataset, and the generator trains on it to produce realistic music. The discriminator then classifies the output as real or fake, and by how much. This is chaaracterised as the loss function. The generator adjusts it's weights using this, and produces better music until the discriminator can't tell them apart anymore. Then, using generator output and real tracks, we retrain the discriminator.

We use a special type of GAN called the Wasserstein GAN with Gradient Penalty (or WGAN-GP) to generate music. While the underlying architecture of a WGAN-GP is very similar to vanilla variants of GAN, WGAN-GPs help overcome some of the commonly seen defects in GANs such as the vanishing gradient problem and mode collapse (see appendix for more details).

When using the Wasserstein loss function, we should train the critic to converge to ensure that the gradients for the generator update are accurate. This is in contrast to a standard GAN, where it is important not to let the critic get too strong, to avoid vanishing gradients.

Therefore, using the Wasserstein loss removes one of the key difficulties of training GANs—how to balance the training of the discriminator and generator. With WGANs, we can simply train the critic several times between generator updates, to ensure it is close to convergence. A typical ratio used is five critic updates to one generator update.

"Babysitting" the learning process

Given that training these models can be an investment in time and resources, we must to continuously monitor training in order to catch and address anomalies if/when they occur. Here are some things to look out for:

What should the losses look like?

The adversarial learning process is highly dynamic and high-frequency oscillations are quite common. However if either loss (critic or generator) skyrockets to huge values, plunges to 0, or get stuck on a single value, there is likely an issue somewhere.

Is my model learning?

Monitor the critic loss and other music quality metrics (if applicable). Are they following the expected trajectories?

Monitor the generated samples (piano rolls). Are they improving over time? Do you see evidence of mode collapse? Have you tried listening to your samples?

How do I know when to stop?

- If the samples meet your expectations
- Critic loss no longer improving
- The expected value of the musical quality metrics converge to the corresponding expected value of the same metric on the training data

In this implementation, we build the generator following a simple four-level Unet architecture by combining _conv2ds and _deconv2d, where _conv2d compose the contracting path and _deconv2d forms the expansive path.

```
def build_generator(condition_input_shape=(32, 128, 1), filters=64,
                    instruments=4, latent_shape=(2, 8, 512)):
    """Buld Generator"""
    c_input = tf.keras.layers.Input(shape=condition_input_shape)
    z_input = tf.keras.layers.Input(shape=latent_shape)

    d1 = _conv2d(c_input, filters, bn=False)
    d2 = _conv2d(d1, filters * 2)
    d3 = _conv2d(d2, filters * 4)
    d4 = _conv2d(d3, filters * 8)
    d4 = tf.keras.layers.Concatenate(axis=-1)([d4, z_input])

    u4 = _deconv2d(d4, d3, filters * 4)
    u5 = _deconv2d(u4, d2, filters * 2)
    u6 = _deconv2d(u5, d1, filters)

    u7 = tf.keras.layers.UpSampling2D(size=2)(u6)
    output = tf.keras.layers.Conv2D(instruments, kernel_size=4, strides=1,
                                    padding='same', activation='tanh')(u7)   # 3   generator = tf.keras.models.Model
([c_input, z_input], output, name='Generator')
    return generator
```

Since the critic tries to classify data as "real" or "fake", it is not very different from commonly used binary classifiers. We use a simple architecture for the critic, composed of four convolutional layers and a dense layer at the end.

```
def _build_critic_layer(layer_input, filters, f_size=4):
    """
    This layer decreases the spatial resolution by 2:

        input:  [batch_size, in_channels, H, W]
        output: [batch_size, out_channels, H/2, W/2]
    """
    d = tf.keras.layers.Conv2D(filters, kernel_size=f_size, strides=2,
                               padding='same')(layer_input)
    # Critic does not use batch-norm
    d = tf.keras.layers.LeakyReLU(alpha=0.2)(d)
    return d


def build_critic(pianoroll_shape=(32, 128, 4), filters=64):
    """WGAN critic."""

    condition_input_shape = (32,128,1)
    groundtruth_pianoroll = tf.keras.layers.Input(shape=pianoroll_shape)
    condition_input = tf.keras.layers.Input(shape=condition_input_shape)
    combined_imgs = tf.keras.layers.Concatenate(axis=-1)([groundtruth_pianoroll, condition_input])
```

```
    d1 = _build_critic_layer(combined_imgs, filters)
    d2 = _build_critic_layer(d1, filters * 2)
    d3 = _build_critic_layer(d2, filters * 4)
    d4 = _build_critic_layer(d3, filters * 8)

    x = tf.keras.layers.Flatten()(d4)
    logit = tf.keras.layers.Dense(1)(x)

    critic = tf.keras.models.Model([groundtruth_pianoroll,condition_input], logit,
                                   name='Critic')


    return critic
```

Training ¶

We train our models by searching for model parameters which optimize an objective function. For our WGAN-GP, we have special loss functions that we minimize as w alternate between training our generator and critic networks:

Generator Loss:

- We use the Wasserstein (Generator) loss function which is negative of the Critic Loss function. The generator is trained to bring the generated pianoroll as close to the real pianoroll as possible.
  - $\frac{1}{m}\sum_{i=1}^{m} - D_w(G(z^i|c^i)|c^i)$

Critic Loss:

- We begin with the Wasserstein (Critic) loss function designed to maximize the distance between the real piano roll distribution and generated (fake) piano roll distribution.
  - $\frac{1}{m}\sum_{i=1}^{m}[D_w(G(z^i|c^i)|c^i) - D_w(x^i|c^i)]$
- We add a gradient penalty loss function term designed to control how the gradient of the critic with respect to its input behaves. This makes optimization of the generator easier.
  - $\frac{1}{m}\sum_{i=1}^{m}(\| \nabla_{\hat{x}^i} D_w(\hat{x}^i|c^i) \|_2 - 1)^2$

References

14. Normal Probability Distributions. (n.d.). Retrieved from Interactive Mathematics: https://www.intmath.com/counting-probability/14-normal-probability-distribution.php

Amazon. (n.d.). *Basics of generative AI and GANs .* Retrieved from https://d32g4xocucupjo.cloudfront.net/

Amazon. (n.d.). *Introduction to autoregressive convolutional neural networks (AR-CNNs).* Retrieved from https://console.aws.amazon.com/deepcomposer/home?region=us-east-1#learningCapsules/autoregressive

Amazon. (n.d.). *Introduction to generative adversarial networks (GANs).* Retrieved from https://console.aws.amazon.com/deepcomposer/home?region=us-east-1#learningCapsules/introToGANs

Class and Instance Attributes. (n.d.). Retrieved from https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556

Class and Instance Attributes. (n.d.). Retrieved from https://www.python-course.eu/python3_class_and_instance_attributes.php

Laerd. (n.d.). *How to do Normal Distributions Calculations.* Retrieved from https://statistics.laerd.com/statistical-guides/normal-distribution-calculations.php

Mathematics, I. (n.d.). *14. Normal Probability Distributions.* Retrieved from https://www.intmath.com/counting-probability/14-normal-probability-distribution.php

Mixins for Fun and Profit. (n.d.). Retrieved from https://easyaspython.com/mixins-for-fun-and-profit-cb9962760556

Python, R. (n.d.). *Primer on Python Decorators.* Retrieved from https://realpython.com/primer-on-python-decorators/

Python, R. (n.d.). *Python's Instance, Class, and Static Methods Demystified.* Retrieved from https://realpython.com/instance-class-and-static-methods-demystified/

What makes sets faster than lists? (n.d.). Retrieved from https://stackoverflow.com/questions/8929284/what-makes-sets-faster-than-lists/8929445#8929445

Wikipedia. (n.d.). *Binomial distribution.* Retrieved from https://en.wikipedia.org/wiki/Binomial_distribution

Wikipedia. (n.d.). *Normal distribution.* Retrieved from https://en.wikipedia.org/wiki/Normal_distribution

Wikipedia. (n.d.). *Probability density function.* Retrieved from https://en.wikipedia.org/wiki/Probability_density_function

Wikipedia. (n.d.). *Standard deviation.* Retrieved from https://en.wikipedia.org/wiki/Standard_deviation

Wikipedia. (n.d.). *Variance.* Retrieved from https://en.wikipedia.org/wiki/Variance