A Generative Approach to Classification

A Simple Explanation

Rahul Agarwal
Aug 25 · 5 min read ★

Image by Prashant Sharma from Pixabay

I always get confused whenever someone talks about generative vs. discriminative classification models.

I end up reading it again and again, yet somehow it eludes me.

So I thought of writing a post on it to improve my understanding.

This post is about understanding Generative Models and how they differ from Discriminative models.

In the end, we will create a simple generative model ourselves.

. . .

Discriminative vs. Generative Classifiers

Problem Statement: Having some input data, `x` we want to classify the data into labels `y`.

A generative model learns the **joint** probability distribution `p(x,y)` while a discriminative model learns the **conditional** probability distribution `p(y|x)`

So really, what is the difference? They both look pretty much the same.

Suppose we have a small sample of data:

```
(x,y) : [(0,1), (1,0), (1,0), (1, 1)]
```

Then `p(x,y)` is

	y=0	y=1
x=0	0	0.25
x=1	0.50	0.25

While `p(y|x)` is

	y=0	y=1
x=0	0	1
x=1	0.67	0.33

As you can see, they model different probabilities.

The discriminative distribution `p(y|x)` could be used straightforward to classify an example `x` into a class `y`. An example of a discriminative classification model is Logistic regression, where we try to model P(y|X).

$$P(Y=1|X=x) = \frac{1}{1+e^{-z}}$$

Logistic Regression

Generative algorithms model `p(x,y)`. An example is the Naive Bayes model in which we try to model P(X,y) and then use the Bayes equation to predict.

. . .

The Central Idea Behind Generative Classification

- Fit each class separately with a probability distribution.
- To classify a new point, find out which distribution is it most probable to come from.

Don't worry if you don't understand yet. You will surely get it by the end of this post.

. . .

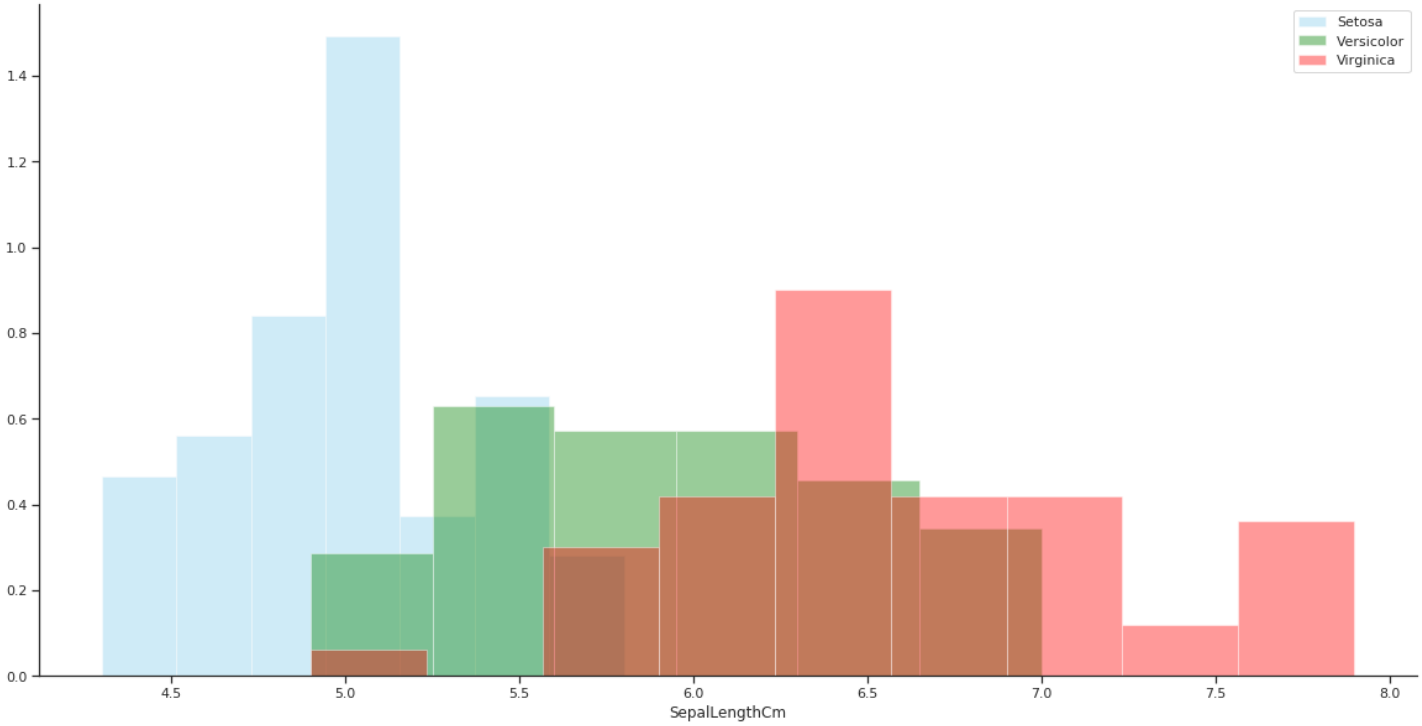
A Small Example

Let us work with the iris dataset.

For our simple example, we will work with a single x variable SepalLength and our target variable Species.

Let us see the distribution of sepal length with Species. I am using plotly_express for this.

```
import plotly_express as px
px.histogram(iris, x = 'SepalLengthCm',color = 'Species',nbins=20)
```



To create generative models, we need to find out two sets of values:

1. Probability of individual classes:

To get individual class probability is fairly trivial- For example, the number of instances in our dataset, which is setosa divided by the total number of cases in the dataset.

```
p_setosa = len(iris[iris['Species']=='Iris-setosa'])/len(iris)
p_versicolor = len(iris[iris['Species']=='Iris-versicolor'])/len(iris)
p_virginica = len(iris[iris['Species']=='Iris-virginica'])/len(iris)
print(p_setosa,p_versicolor,p_virginica)
```

probs.py hosted with ♡ by GitHub view raw

```
0.3333333333333333 0.3333333333333333 0.3333333333333333
```

The iris dataset is pretty much balanced.

2. The probability distribution of x for each class:

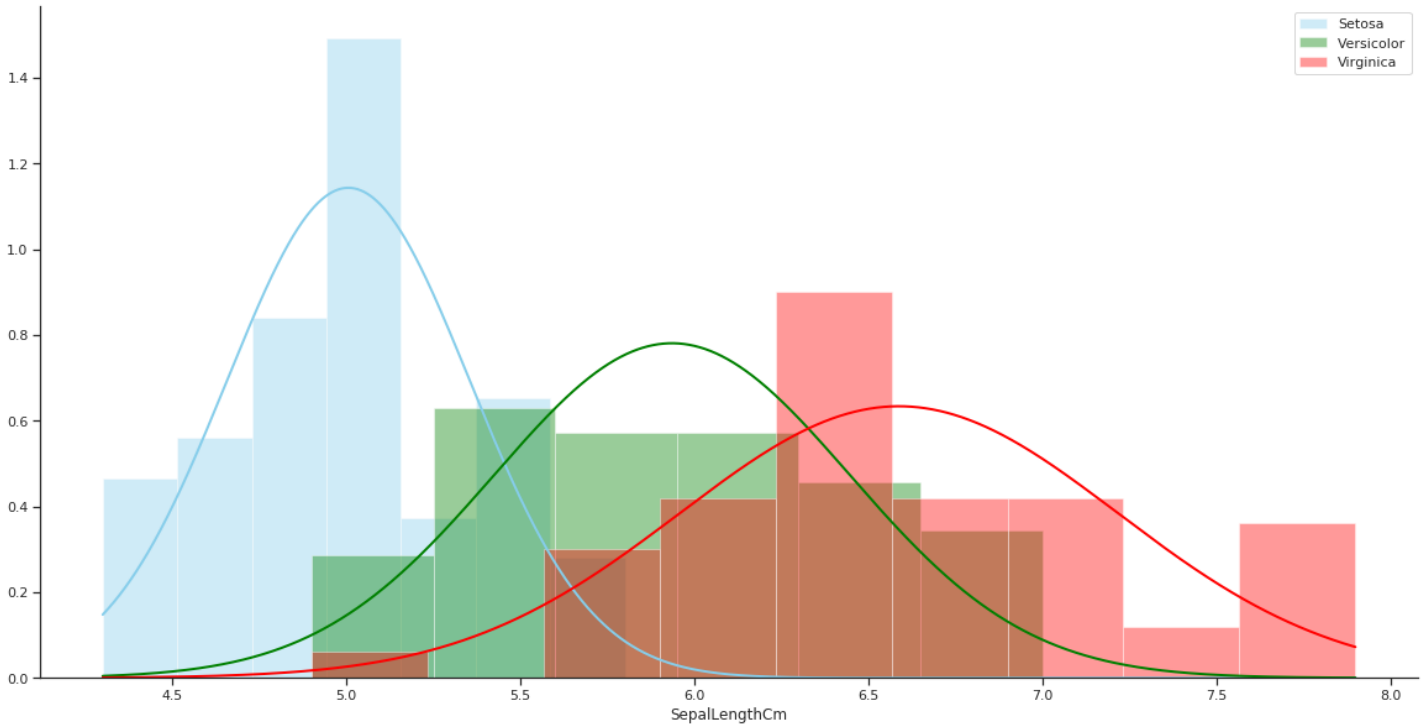
Here we fit a probability distribution over our X. We assume here that the X data is distributed normally. And hence we can find the sample means and variance for these three distributions(As we have three classes)

```
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt

sns.set(style="ticks")
# calculate the pdf over a range of values
xx = np.arange(min(iris['SepalLengthCm']), max(iris['SepalLengthCm']),0.001)
x = iris[iris['Species']=='Iris-setosa']['SepalLengthCm']
sns.distplot(x, kde = False, norm_hist=True,color='skyblue',label = 'Setosa')
yy = stats.norm.pdf(xx,loc=np.mean(x),scale=np.std(x))
plt.plot(xx, yy, 'skyblue', lw=2)
x = iris[iris['Species']=='Iris-versicolor']['SepalLengthCm']
sns.distplot(x, kde = False, norm_hist=True,color='green',label = 'Versicolor')
yy = stats.norm.pdf(xx,loc=np.mean(x),scale=np.std(x))
plt.plot(xx, yy, 'green', lw=2)
x = iris[iris['Species']=='Iris-virginica']['SepalLengthCm']
g = sns.distplot(x, kde = False, norm_hist=True,color='red',label = 'Virginica')
yy = stats.norm.pdf(xx,loc=np.mean(x),scale=np.std(x))
plt.plot(xx, yy, 'red', lw=2)
sns.despine()
g.figure.set_size_inches(20,10)
```

```
23  g.legend()
```

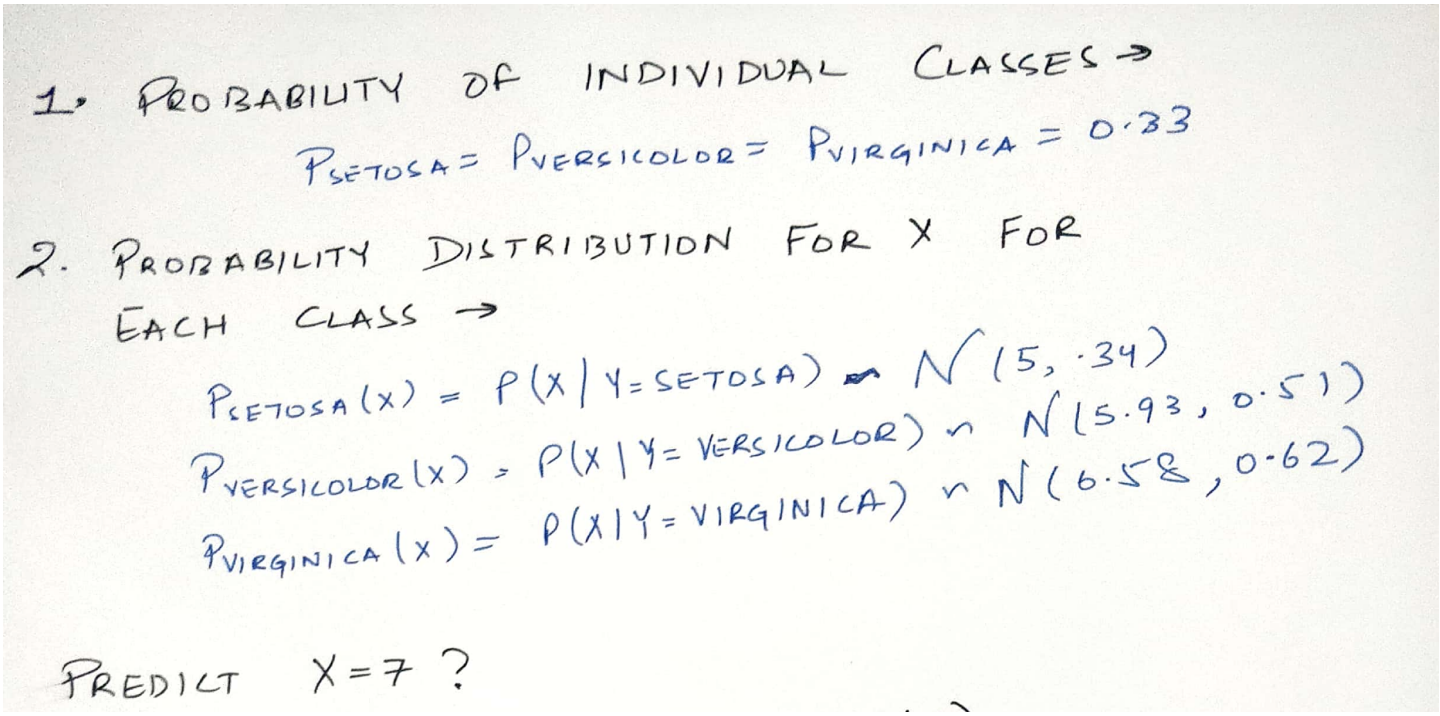
plot.py hosted with ♡ by GitHub view raw

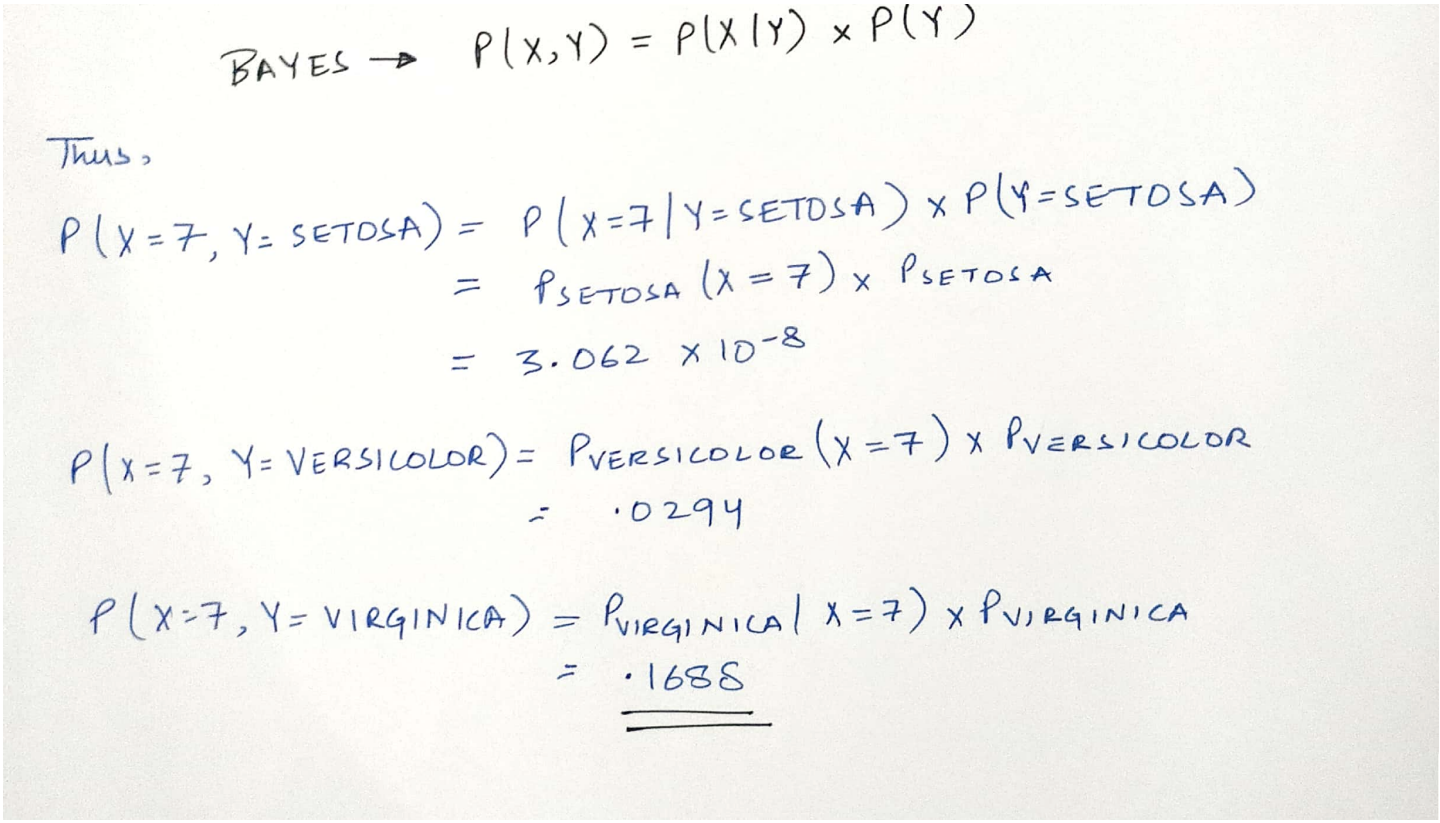


In the above graph, I have fitted three normal distributions for each of the species just using sample means and variances for each of the three species.

So, how do we predict using this?

Let us say we get a new example with SepalLength = 7 cm.





Since we see that the maximum probability comes for Virginica, we predict virginica for x=7, and based on the graph too; it looks pretty much the right choice.

You can get the values using the code too.

```
x = iris[iris['Species']=='Iris-setosa']['SepalLengthCm']
print("Setosa",stats.norm.pdf(7,loc=np.mean(x),scale=np.std(x))*.33)
x = iris[iris['Species']=='Iris-versicolor']['SepalLengthCm']
print("Versicolor",stats.norm.pdf(7,loc=np.mean(x),scale=np.std(x))*.33)
x = iris[iris['Species']=='Iris-virginica']['SepalLengthCm']
print("Virginica",stats.norm.pdf(7,loc=np.mean(x),scale=np.std(x))*.33)
```

calculation.py hosted with ♡ by GitHub view raw

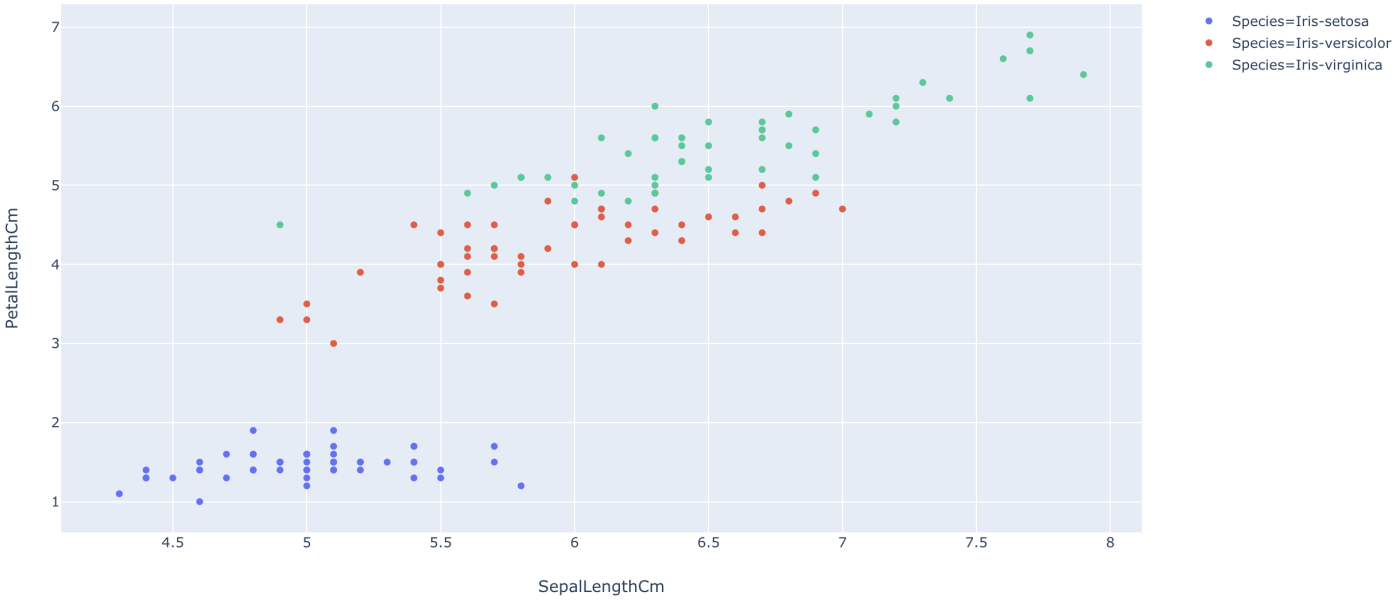
```
Setosa 3.062104211904799e-08
Versicolor 0.029478757465669376
Virginica 0.16881724812694823
```

. . .

This is all well and good. But when do we ever work with a single variable?

Let us extend our example for two variables. This time let us use PetalLength too.

```
px.scatter(iris, 'SepalLengthCm', 'PetalLengthCm',color = 'Species')
```



So how do we proceed in this case?

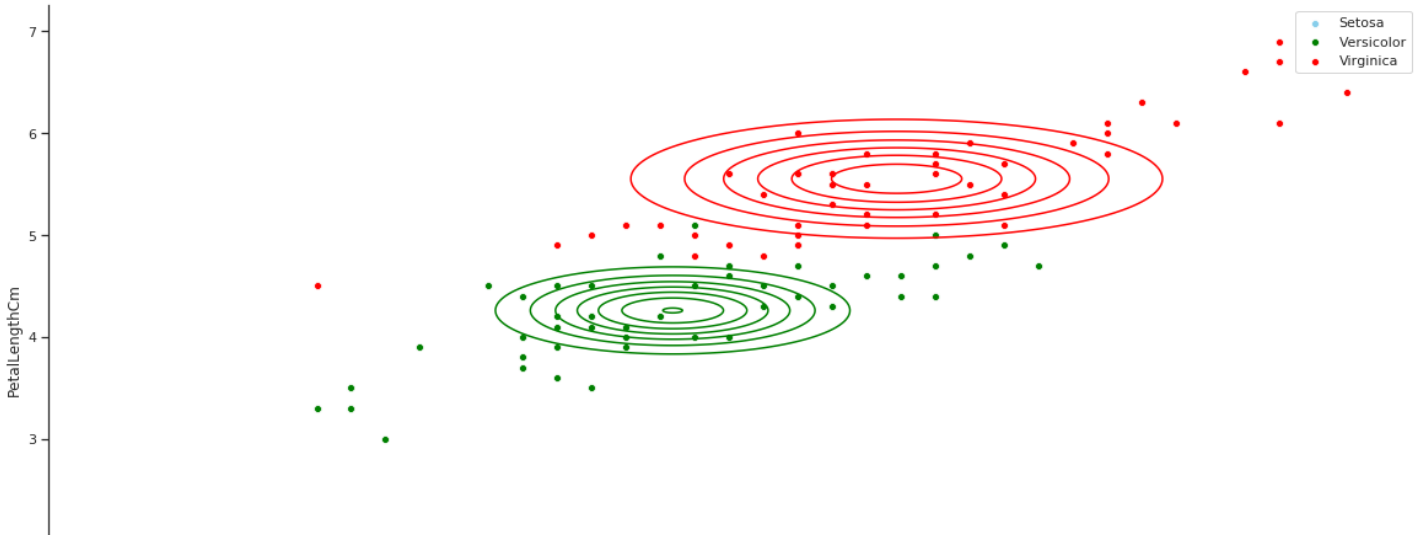
The first time we had fit a Normal Distribution over our single x, this time we will fit Bivariate Normal.

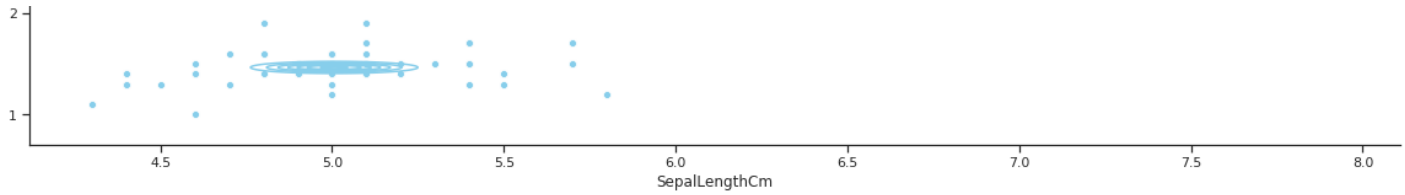
```
import numpy as np
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt
from matplotlib.mlab import bivariate_normal
sns.set(style="ticks")
# SETOSA
x1 = iris[iris['Species']=='Iris-setosa']['SepalLengthCm']
x2 = iris[iris['Species']=='Iris-setosa']['PetalLengthCm']
sns.scatterplot(x1,x2, color='skyblue',label = 'Setosa')
mu_x1=np.mean(x1)
mu_x2=np.mean(x2)
sigma_x1=np.std(x1)**2
sigma_x2=np.std(x2)**2
xx = np.arange(min(x1), max(x1),0.001)
yy = np.arange(min(x2), max(x2),0.001)
X, Y = np.meshgrid(xx, yy)
Z = bivariate_normal(X,Y, sigma_x1, sigma_x2, mu_x1, mu_x2)
```

```
plt.contour(X,Y,Z,colors='skyblue')
# VERSICOLOR
x1 = iris[iris['Species']=='Iris-versicolor']['SepalLengthCm']
x2 = iris[iris['Species']=='Iris-versicolor']['PetalLengthCm']
sns.scatterplot(x1,x2,color='green',label = 'Versicolor')
mu_x1=np.mean(x1)
mu_x2=np.mean(x2)
sigma_x1=np.std(x1)**2
sigma_x2=np.std(x2)**2
xx = np.arange(min(x1), max(x1),0.001)
yy = np.arange(min(x2), max(x2),0.001)
X, Y = np.meshgrid(xx, yy)
Z = bivariate_normal(X,Y, sigma_x1, sigma_x2, mu_x1, mu_x2)
plt.contour(X,Y,Z,colors='green')
# VIRGINICA
x1 = iris[iris['Species']=='Iris-virginica']['SepalLengthCm']
x2 = iris[iris['Species']=='Iris-virginica']['PetalLengthCm']
g = sns.scatterplot(x1, x2, color='red',label = 'Virginica')
mu_x1=np.mean(x1)
mu_x2=np.mean(x2)
sigma_x1=np.std(x1)**2
sigma_x2=np.std(x2)**2
xx = np.arange(min(x1), max(x1),0.001)
yy = np.arange(min(x2), max(x2),0.001)
X, Y = np.meshgrid(xx, yy)
Z = bivariate_normal(X,Y, sigma_x1, sigma_x2, mu_x1, mu_x2)
plt.contour(X,Y,Z,colors='red')
sns.despine()
g.figure.set_size_inches(20,10)
g.legend()
```

plot2.py hosted with ♡ by GitHub view raw

Here is how it looks:





Now the rest of the calculations remains the same.

Just the normal gets replaced by Bivariate normal in the above equations. And as you can see, we get a pretty better separation amongst the classes by using the bivariate normal.

As an extension to this case for multiple variables(more than 2), we can use the multivariate normal distribution.

. . .

Conclusion

Generative models are good at generating data. But at the same time, creating such models that capture the underlying distribution of data is extremely hard.

Generative modeling involves a lot of assumptions, and thus, these models don't perform as well as discriminative models in the classification setting. In the above example also we assumed that the distribution is normal, which might not be correct and hence may induce a bias.

But understanding how they work is helpful all the same. One class of such models is called generative adversarial networks which are pretty useful for generating new images and are pretty interesting too.

Here is the kernel with all the code along with the visualizations.

. . .

If you want to learn more about generative models and Machine Learning, I would recommend this Machine Learning Fundamentals course from the University of San Diego. The above post is by and large inspired from content from this course in the MicroMasters from SanDiego, which I am currently working on to structure my Data Science learning.

. . .

Thanks for the read. I am going to be writing more beginner-friendly posts in the future too. Follow me up at **Medium** or Subscribe to my **blog** to be informed about them. As always, I welcome feedback and constructive criticism and can be reached on Twitter @mlwhiz.