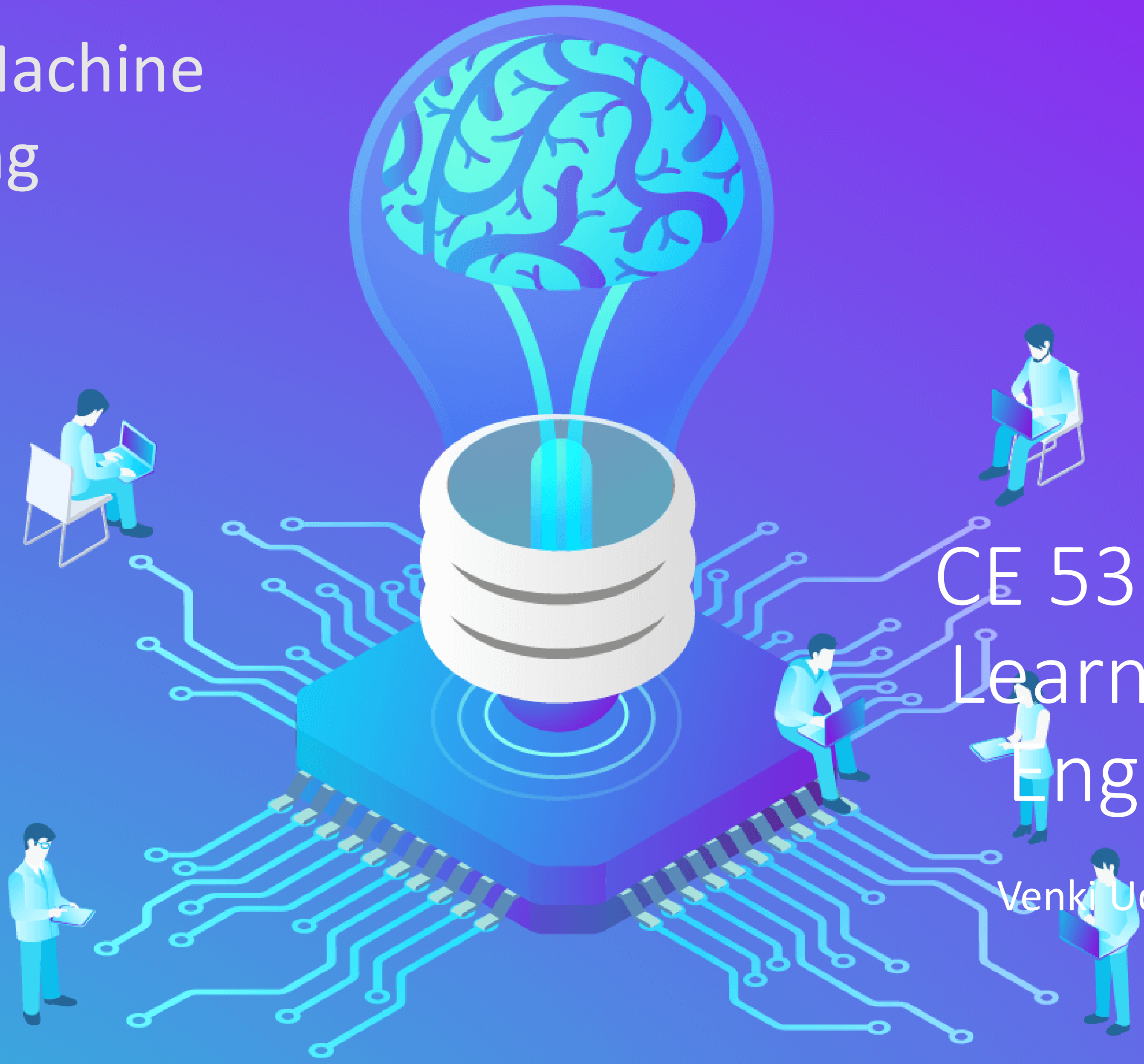
Python for Machine Learning

Naïve Bayes



CE 5331 Machine Learning for Civil Engineers

Venki Uddameri, Ph.D. , P.E.

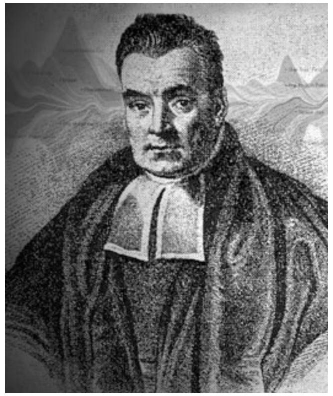
Recap

- What is Machine Learning
- How is it useful for Civil Engineers
- Overview of Machine Learning Methods
- Linear Regression
  - Bivariate
  - Regression interpretation
  - Multivariate
- Logistic Regression
  - Maximum likelihood estimation
  - Regularization (introduction)

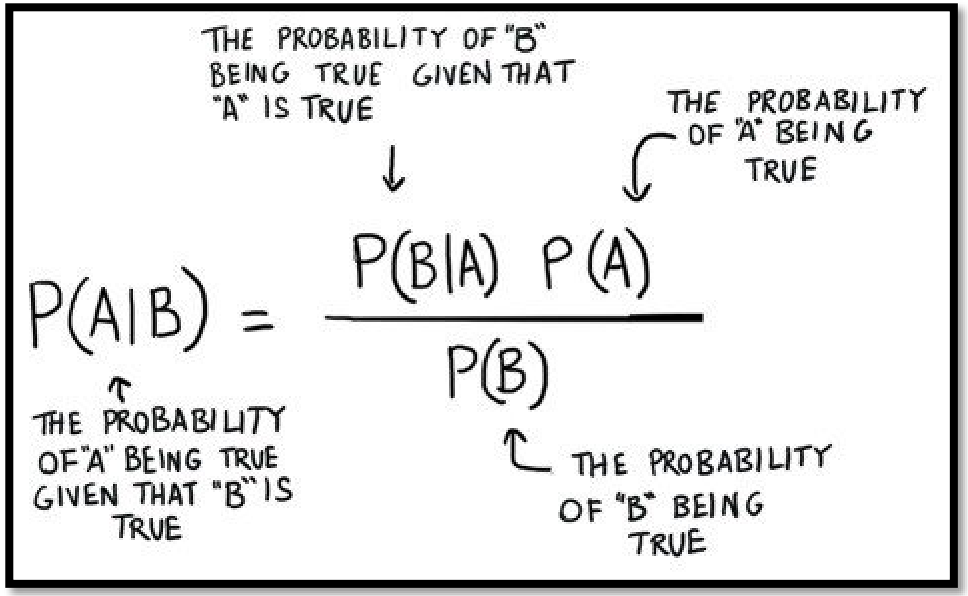
Python – Introduction
Python – Functions
Python - Pandas
Python – np, scipy, statsmodels
Python – Scikit learn – linear, metrics
Python – Matplotlib, seaborn

In this module we shall look at Lazy and Easy Learners

Naïve Bayes



- Naïve Bayes classifiers use Bayes' theorem to perform classification
- Naïve in Naïve Bayes refers to the simplistic assumption of conditional independence used in the modeling
- Naïve Bayes can be used for both classification and regression problems
    - Seems to work better on classification



While Naïve Bayes is an easy (simple) algorithm it is known to perform surprisingly well in many situations
As Naïve Bayes is a simple model – It is recommended this method be tried as a baseline case
A more complex ML model must be able to perform significantly better than the Naïve Bayes Classifier

Naïve Bayes Classification

- We want to calculate the posterior probability of a class given a set of predictors
- Posterior

$$posterior \propto prior \times likelihood$$

Posterior is conditioned upon X

Likelihood is computed from the data

What we want to know; the Posterior probability of Class j given a predictor x

The Likelihood; the probability of the predictor given a Class j. Its computed from the training-set.

The Prior probability of Class j; what we know about the class distribution before we consider x.

$$P(Class_j \mid x) = \frac{P(x \mid Class_j) \times P(Class_j)}{P(x)}$$

The Evidence. In practice, there's interest only in the numerator (denominator is effectively constant)

Applying the independence assumption

$$P(x \mid Class_j) = P(x_1 \mid Class_j) \times P(x_2 \mid Class_j) \times \ldots \times P(x_k \mid Class_j)$$

Substituting the independence assumption, we derive the Posterior probability of Class j given a new instance x' as...

$$P(Class_j \mid x') = P(x'_1 \mid Class_j) \times P(x'_2 \mid Class_j) \times \ldots \times P(x'_k \mid Class_j) \times P(Class_j)$$

Naïve Bayes

- Naïve Bayes has several advantages
  - Works with relatively small datasets
  - Is easier to implement compared to many methods
  - Training these classifiers are very fast
  - Can used in an adaptive mode
    - Retrain the model as new data become available
  - Naïve Assumption helps overcome the ‘curse of dimensionality’ associated with Bayesian modeling
  - Can work with a variety of different data types
    - We need to (and can) specify various probability distribution functions

Parameter Estimation - NB

- We divide the dataset into two modes
    - Training
        - Use to compute the prior and likelihood
    - Testing
        - Use to evaluate the prediction capabilities
- We need to specify the conditional probability
    - How output y varies with an input x
- Naïve Bayes assumes the X variables are independent
    - So the joint probability is a multiplication of the individual probabilities

Naïve Bayes Classification in Python

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

for all i, this relationship is simplified to

Naïve Assumption

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Denominator is only a normalizing constant so calculation is not necessary for classification

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

Naïve Bayes Classification - Python

- Scikit learn can fit many types of naïve Bayes classifiers
  - Naïve Bayesian classifiers differ based on the underlying probability function used for the likelihood function

1.9. Naive Bayes

1.9.1. Gaussian Naive Bayes — Continuous Likelihood

1.9.2. Multinomial Naive Bayes

1.9.3. Complement Naive Bayes — Imbalanced datasets

1.9.4. Bernoulli Naive Bayes

1.9.5. Categorical Naive Bayes

1.9.6. Out-of-core naive Bayes model fitting

Discrete Likelihood

Very large datasets

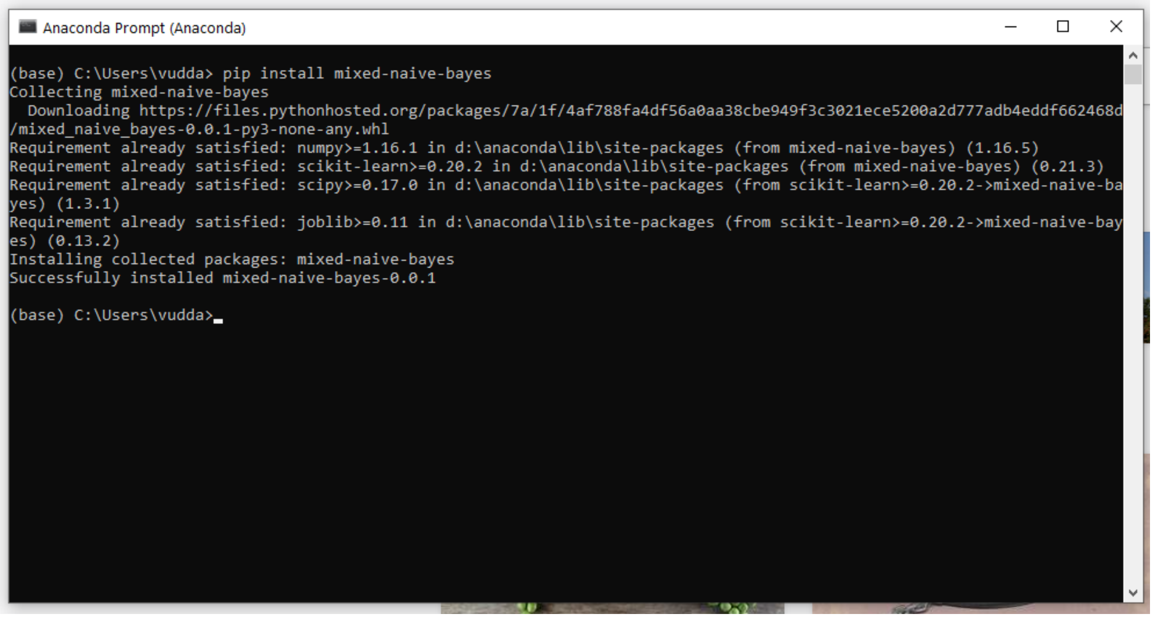
$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The parameters σ_y and μ_y are estimated using maximum likelihood.

https://scikit-learn.org/stable/modules/naive_bayes.html

Naïve Bayes

- Scikit learn cannot mix categorical and continuous independent variables
  - You can fit two models and then multiply the probabilities
  - This is because of the condition assumption of the Naïve Bayes
  - T
- There is a package called Mixed Naïve Bayes
  - Written Numpy
  - Available on git hub and PIP - https://pypi.org/project/mixed-naive-bayes/
  - Uses Mixed Bernoulli and Gaussian for categorical and discrete variables



- R package – ‘naivebayes’ also works well

Illustrative Example

- Predicting damage to culverts in Texas

Satisfactory

Unsatisfactory

Code	Meaning	Description
9	Excellent	As new
8	Very Good	No problems noted.
7	Good	Some minor problems.
6	Satisfactory	Structural elements show some minor deterioration.
5	Fair	All primary structural elements are sound but may have minor section loss, cracking, spalling or scour.
4	Poor	Advanced section loss, deterioration, spalling or scour.
3	Serious	Loss of section, deterioration, spalling or scour has seriously affected primary structural components. Local failures are possible. Fatigue cracks in steel or shear cracks in concrete may be present.
2	Critical	Advanced deterioration of primary structural elements. Fatigue cracks in steel or shear cracks in concrete may be present or scour may have removed substructure support. Unless closely monitored it may be necessary to close the bridge until corrective action is taken.
1	Imminent Failure	Major deterioration or section loss present in critical structural components or obvious vertical or horizontal movement affecting structure stability. Bridge is closed to traffic but with corrective action may put back in light service.
0	Failed	Out of service, beyond corrective action.

Source: United States Department of Transportation. Recording and Coding Guide for the Structure Inventory and Appraisal of the Nation's Bridges. Washington, D.C., 1995, page 38.

Data

- Data was taken from Federal Highway Administration (FHWA)
  - https://www.fhwa.dot.gov/bridge/nbi/ascii.cfm
  - Year 2018 data release (updated 4/22/2019)
  - Downloaded csv file and cleaned up the dataset
- Previous studies indicate
  - Reconstruction Record (1 = Reconstructed; 0 = No Reconstruction)
  - Age (Inspection Date – (re)Construction Year)
  - Age^2
  - ADT (Average Daily Traffic per lane)
- The % of Truck Traffic on the Culvert could also be important
  - PTRCK
    - A measure of higher loads → greater damage potential

Culvert type could also be important but not considered here due to lack of data

Lu, Pan, Hao Wang, and Denver Tolliver. "Prediction of Bridge Component Ratings Using Ordinal Logistic Regression Model." *Mathematical Problems in Engineering* 2019 (2019). Available online: https://www.hindawi.com/journals/mpe/2019/9797584/

Python Code

- Step 1: Import Libraries
  - Usual suspects
  - import mixed_naive_bayes as mnb
- Step 2: Change Working Directory
  - Use os.chdir
- Step 3: Read the data
  - Use pandas
  - Subset the required variables
  - Add other variables as necessary
- Step 4: Split the data into training and testing
  - Use sklearn train_test_split
  - Use the same seed as before to ensure the same split
- Step 5: Train the model
  - Create a model object
  - Fit the model
  - Make predictions
  - Use mixed_naive_bayes library
- Step 6: Evaluate the model (testing data)
  - Predict testing data
  - Contingency table
  - Accuracy, precision, recall
  - ROC – AUC metric
  - Use sklearn metrics

Step 1: Import Libraries

```
# Import libraries
import os
import numpy as np
import pandas as pd
import mixed_naive_bayes as mnb
from sklearn.model_selection import train_test_split
from sklearn import metrics
import seaborn as sns
from matplotlib import pyplot as plt
```

Step 2: Change Working Directory

```
# Change Working Directory
os.chdir('D:/Dropbox/000CE5333Machine Learning/Week5-LazyEasyLearners/Code')
```

Step 3: Read the Data

```
# Read the dataset
a = pd.read_csv('TXculvertdata.csv') # read our dataset
features = ['SVCYR','ADT','Reconst','PTRUCK'] # INPUT DATA FEATURES
X = a[features] # DATAFRAME OF INPUT FEATURES
SVCYR2 = a['SVCYR']**2  # Add SVCYR square to the dataset
X['SVCYR2'] = SVCYR2  # CALCULATE THE SQUARE OF AGE
Y = a['Culvert_Damage'] # ADD IT TO THE INPUT FEATURE DATAFRAME
```

You can compare the Naïve Bayes Model to the Results from the Logistic Regression Model

Python Code

Step 4: Split training and testing datasets

```
# Split into training and testing data
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.30,random_state=10)
```

Step 5: Train the Model

```
# Naive Bayes Model.  Note Reconst is a categorical variable X[2]
clf = mnb.MixedNB(categorical_features=[2])
clf.fit(X_train,y_train)
clf.predict(X_train)
```

Step 6a: Predict Testing Data

```
# Predict testing data
y_pred=clf.predict(X_test) # predict testing data
yprob = clf.predict_proba(X_test) #output probabilities
```

Step 6b: Create a Contingency Table

```
# Perform evaluation using contingency table
# Create a confusion Matrix
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix # y_test is going be rows (obs), y_pred (predicted) are cols
```

Step 6c: Compute Accuracy, Precision, Recall

```
# Evaluate usng accuracy, precision, recall
print("Accuracy:",metrics.accuracy_score(y_test, y_pred)) # overall accuracy
print("Precision:",metrics.precision_score(y_test, y_pred)) # predicting 0 (Sat)
print("Recall:",metrics.recall_score(y_test, y_pred)) # predicting 1 (unsat)
```

Step 6d: Create AUC curve

```
# ROC Curve
y_pred_proba = clf.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(round(auc,4)))
plt.legend(loc=4)
plt.xlabel('1-Specificity')
plt.ylabel('Sensitivity')
plt.grid() # Plot the grid
plt.show() #show the curve
```

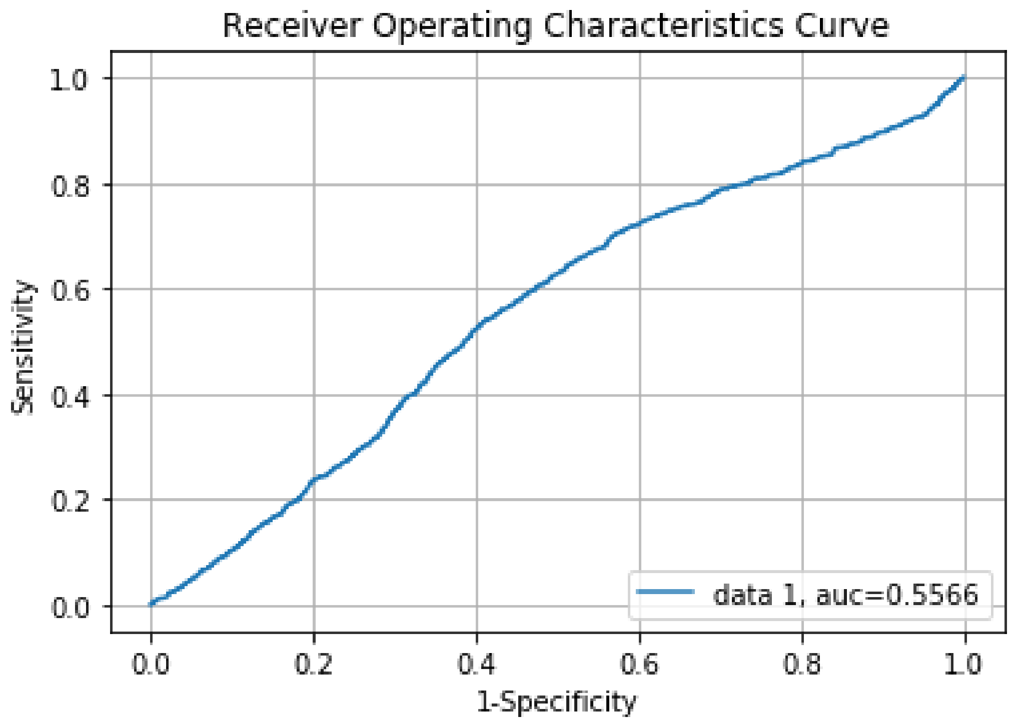
Results

0.663
0.613
0.504

Contingency Table

		Predicted	
		0	1
Observed	0	2633	872
	1	1189	1292

Metric	Value	Remarks
Accuracy	0.656	How well both 0 and 1 are predicted
Precision	0.597	How well 0 states are predicted
Recall	0.520	How well 1 states are predicted



Accuracy and Recall is slightly better than the logistic Regression model

Precision is slightly inferior to the LR model

Model predicts the damage state better than the LR model

You should know

- What is a Naïve Bayes Classifier
- What is the principle behind a Naïve Bayes Classifier
- What is the assumption that makes Naïve Bayesian Classifier – Naïve
- How to set up a Naïve Bayesian Classifier
  - In Python
    - Scikit learn
    - Mixed-naïve-Bayesian library
  - In R
    - ‘naivebayes’ library

Installing mixed_naive_bayes library using pip

Implementing Naïve Bayes Classifier in Python