# Anomaly Detection

Using Python

# “Anomalies” Difficult to Characterize

- Define as “Very Different but Rare”?
  - ⇒ outlier detection
  - Not applicable to security -- persistent threats
- How about “New and Not Normal”?
  - ⇒ novelty detection
  - How “normal” is your definition of normal?
- Is the “Anomaly” Worth Reporting?

# Unique Attributes of the Problem

- Traditional “Detection” Problems:
  - Multi-classification
  - #1: train on labeled data ⇒ label new data
  - #2: cluster unlabeled data ⇒ label new data
- Anomaly Detection:
  - One-class classification
  - No known correct model for “different”
  - Detect them anyway as they arise

# Despite Difficulties, Successful Area

- Fraud Detection, Network Security, Flight Safety, ...

- We’ll Try to Address Previous Points from a Practical Perspective

# Credit Card Fraud Example

- Scenario:
  - Merchant purchases stolen #’s to buy own goods
  - First-time offender
  - Uses all the stolen cards at once

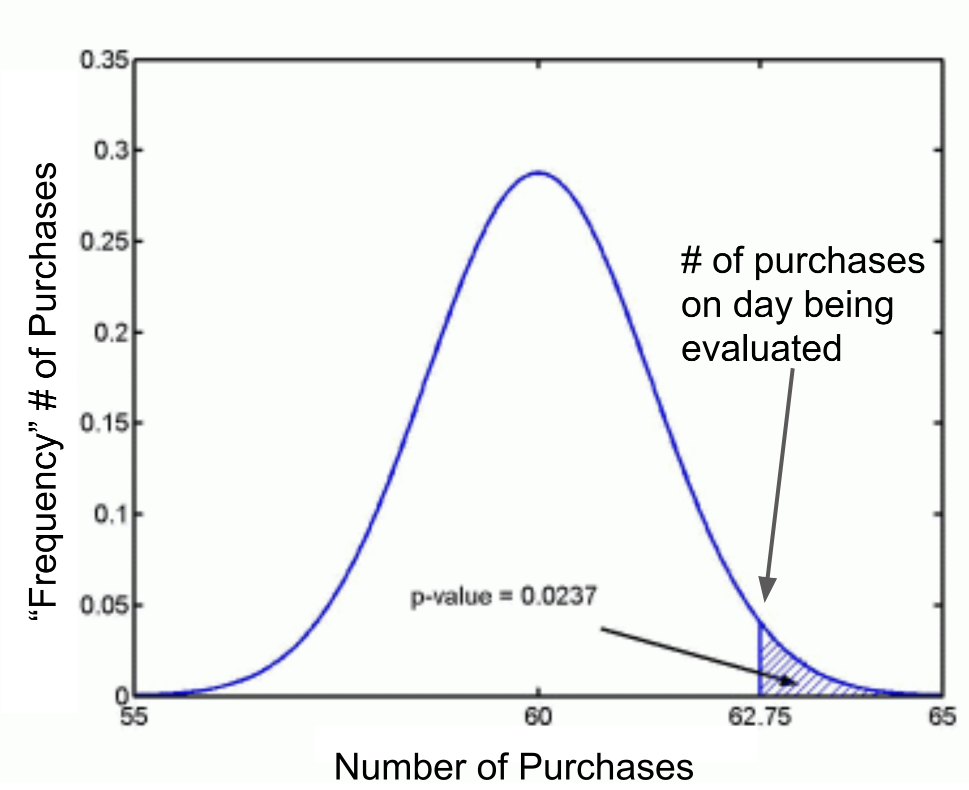
- Our Approach:
  - We have historic data of merchant transaction rates
  - We’ll detect abnormally **high** transaction rates

# Steps in Python

1. Gather purchase history for the merchant
   a. # of purchases per day
   b. # purchases per times of day

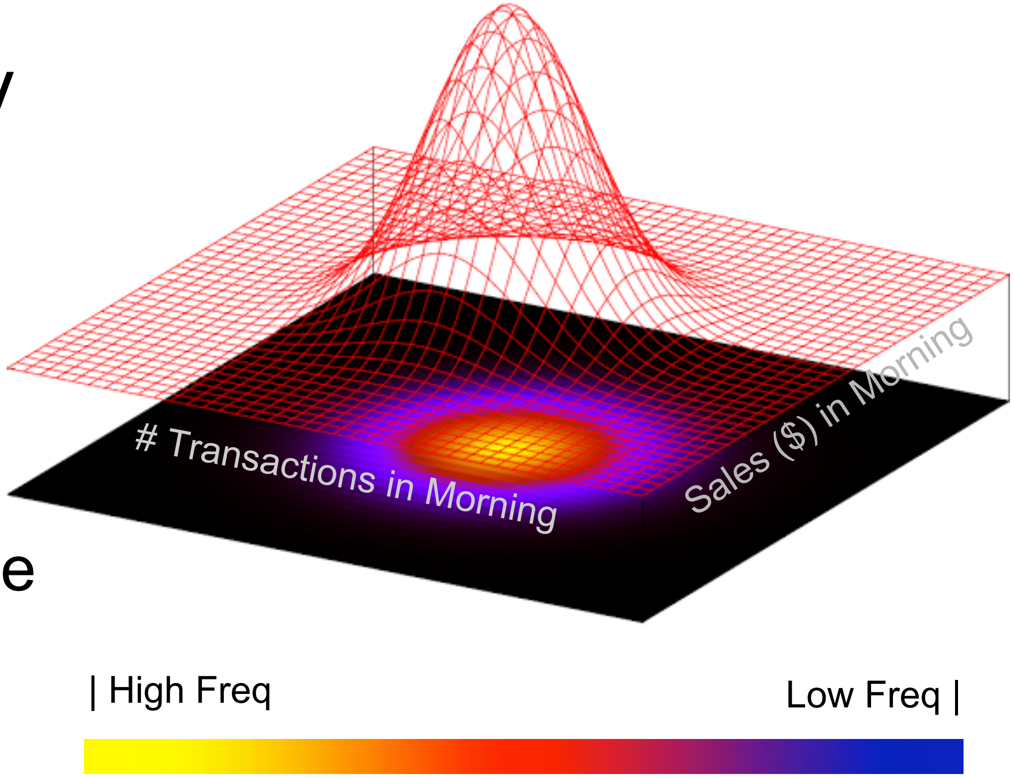
2. Approximate purchase data as Gaussian

3. On new day, determine if anomalously high by computing p-value

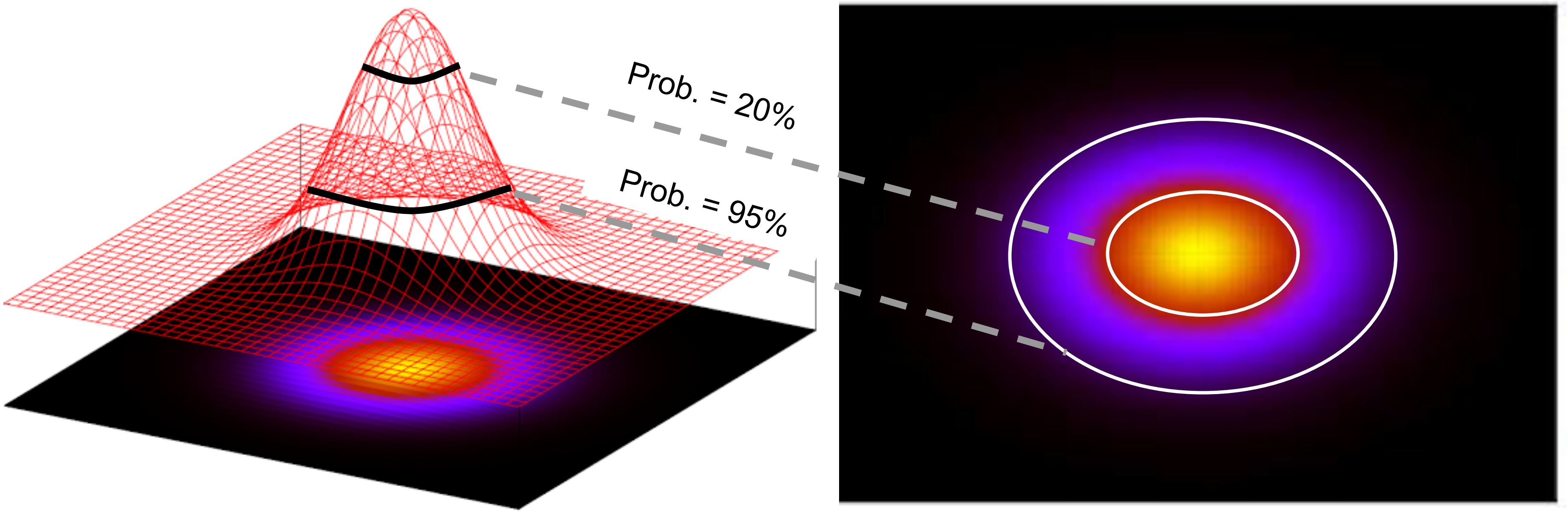


# Two Features Are Easy to Interpret

- Include Total Daily Sales ($)
- Easy to Visualize
  - 3-D Histograms
  - Colorize in the plane



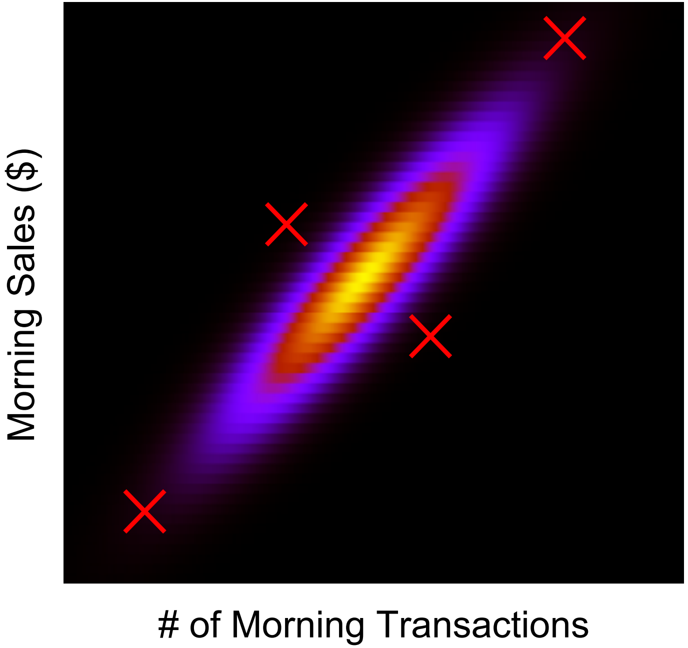
# Levels Sets ⇒ Probabilities



# Which Points Indicate Fraud?

- Same Level Set
  - Level set probability ~ 99%
  - All anomalous

- What Is the Impact of Using Regression?

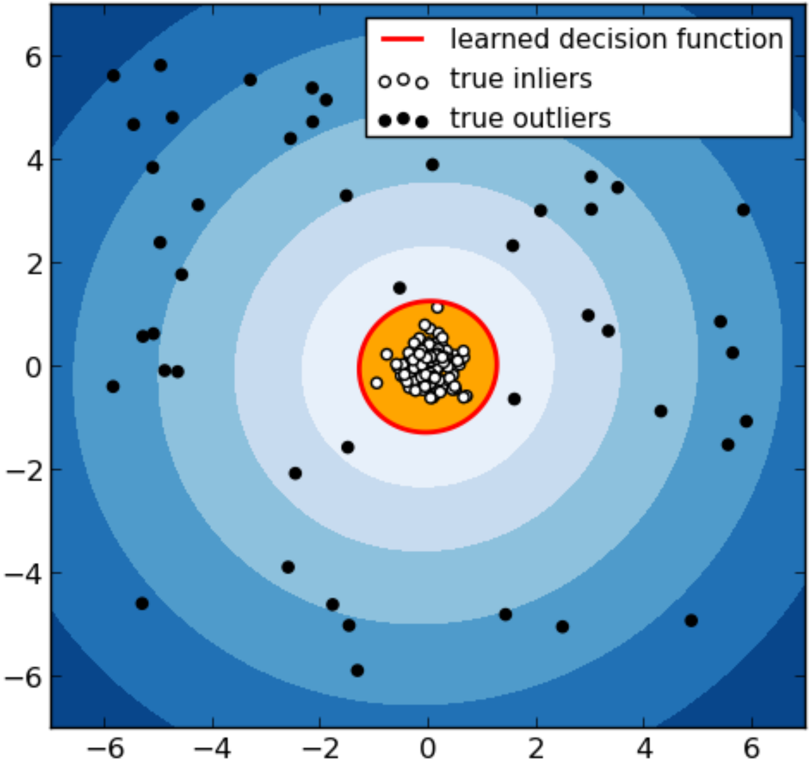


# Robust Distribution Fitting in Python

- Fitting Gaussians in Noisy Data Is Challenging

- Robust: Find Pts that Minimize Area of X% Level Set

- Marks Distant Points as Outliers (Uses Probability)

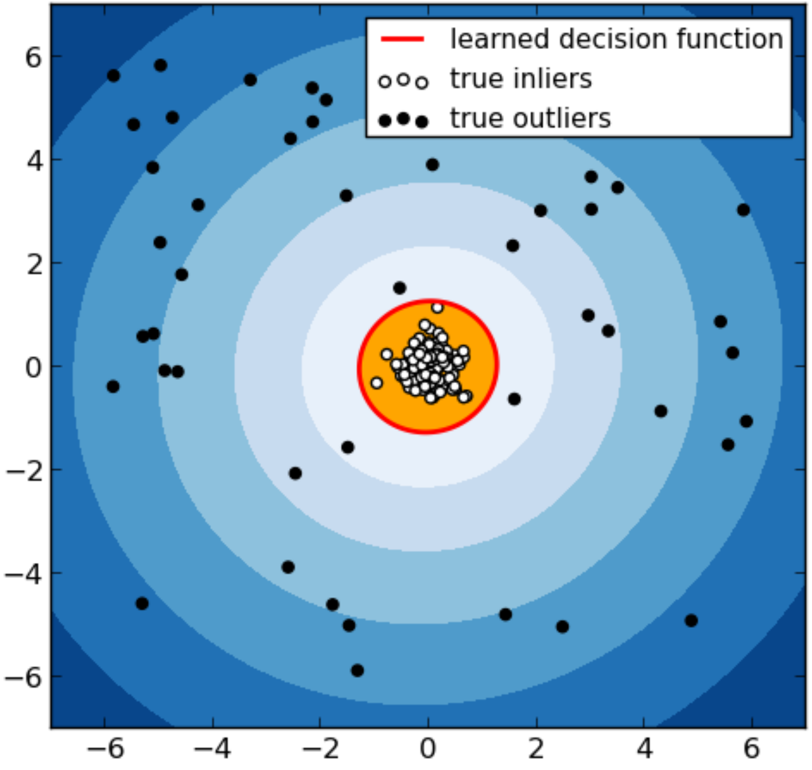


# Credit Card Fraud Example

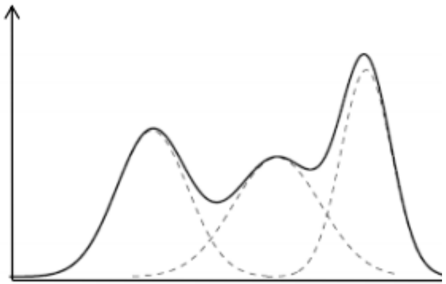
- Scenario:
  - Two merchants purchase stolen #'s to buy goods
  - We lack historic data for the merchants
  - Have data from many other similar merchants
- Our Approach:
  - Perform a robust estimation of # customers + sales
  - Detect fraudulent merchants as an outliers

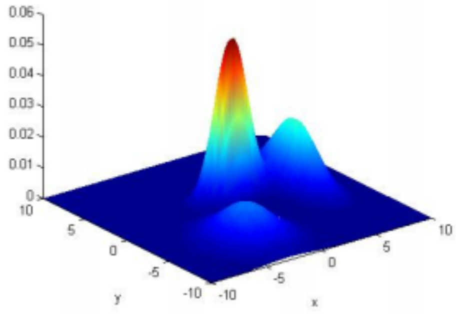
# Steps in Python

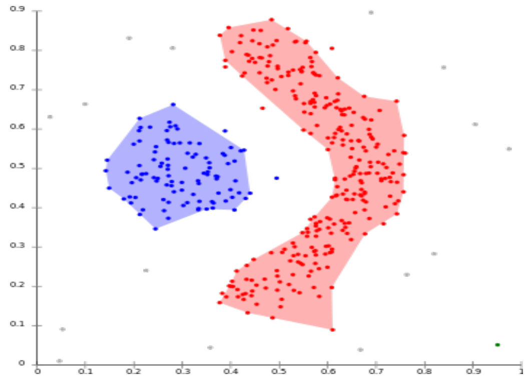
1. Gather sales and transaction data from all merchants
2. Run robust covariance estimation
3. Find those merchants that are outliers



# Non-Gaussian Data Is Typical





Mixture Models:

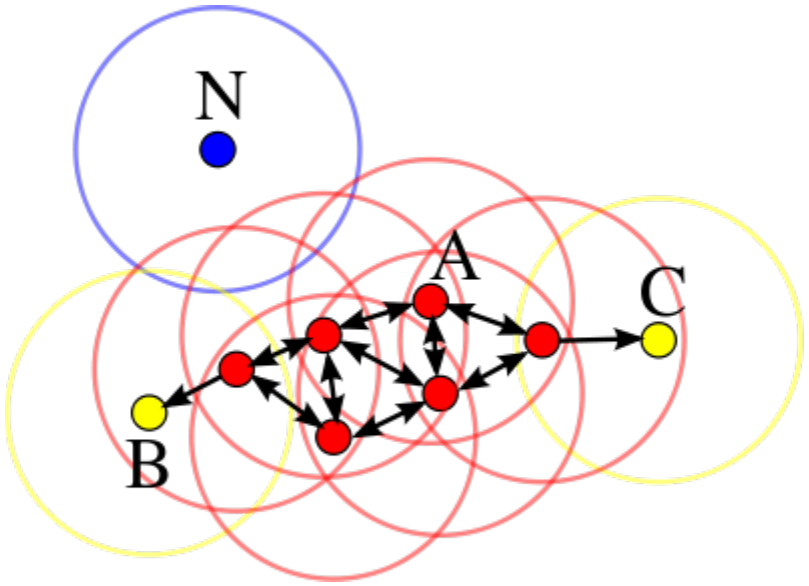
- Probabilistic
- GMMs: sensitive to outliers, need to set K
- Other distributions are robust
- # components can be set by several methods

Distance-Based Methods:

- Non-probabilistic
- Easy-to-understand
- DBSCAN: clusters while searching for outliers

# DBSCAN: Density Based Clustering

- Point Classification:
  - Core: satisfies density
  - Boundary: connected to a core point
  - Noise: not connected to a core point
- Key Is Efficient Implementation



# Credit Card Fraud Example

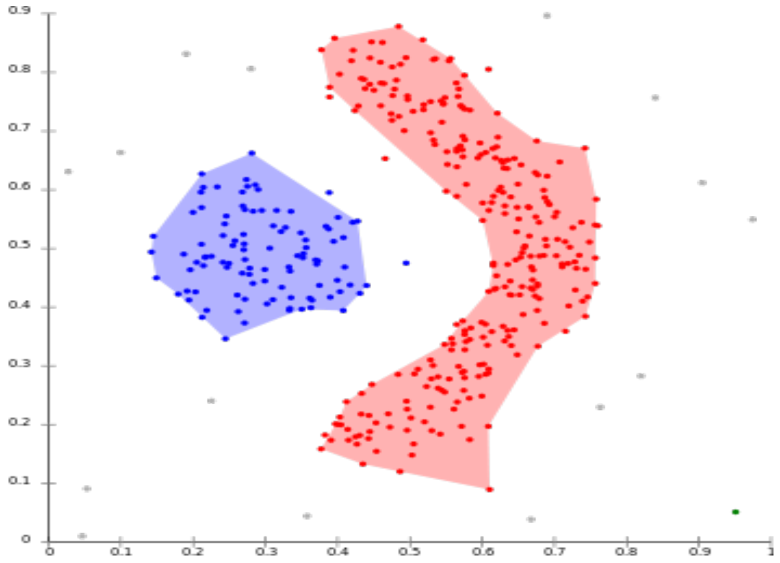
- Scenario:
  - Two merchants purchase stolen #'s to buy goods
  - We lack historic data for the merchants
  - Have data from many other similar merchants
- Our Approach:
  - Apply DBSCAN to # customers + sales
  - Detect fraudulent merchants as an outliers

# Steps in Python

1. Gather sales and transaction data from all merchants

2. Run DBSCAN

3. Find those merchants that are outliers (noise)



# Other Things to Consider...

- Cost of False Positives: What Is the Impact of Accusing a Merchant of Fraud?

- Timescales: Some Anomalies Are Only Visible over Long Periods

- Ground Truth: How Can You Take Advantage of Past True or False Positives?

# Final Remarks

- Successful Anomaly Detection Requires:
  - Knowing or robustly estimating normal behavior
  - Knowing which low-probability events to ignore
  - Knowing what part of your data to examine
- Many, Many Algorithms Exist
  - We focused on one probabilistic approach
  - Distance-based heuristics are also used
  - Not covered: 1-class SVMs and many others...