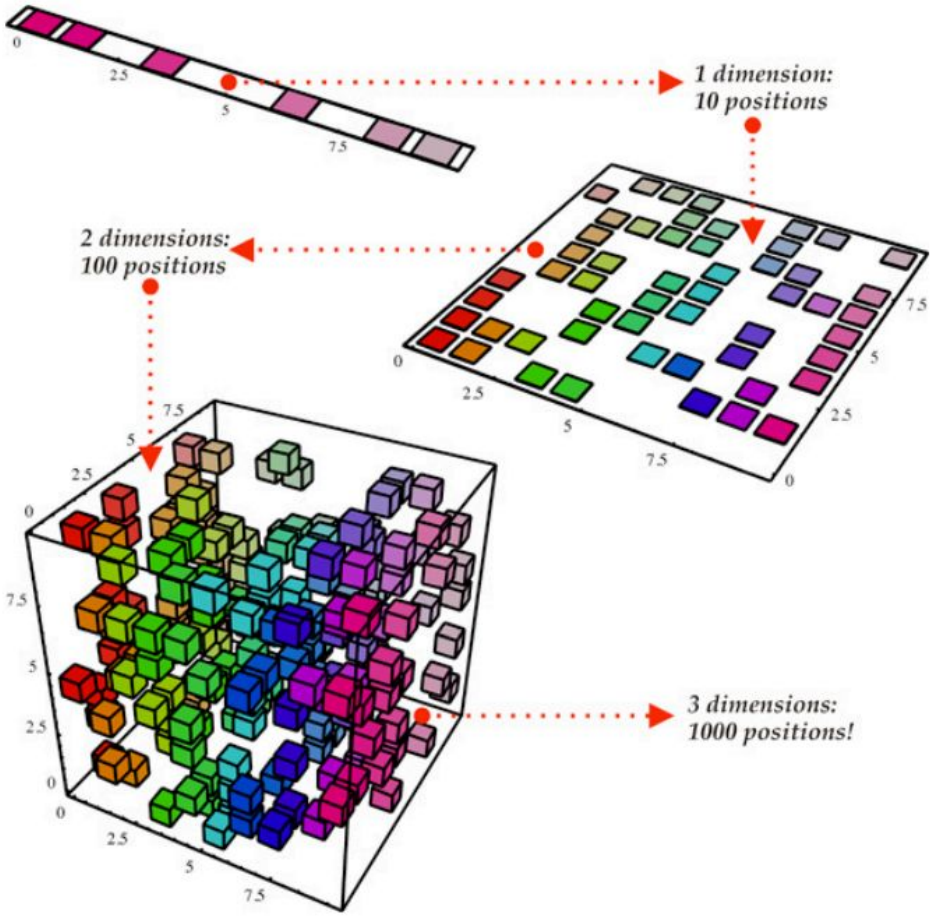
Dimensionality Reduction and Anomaly Detection

The Curse of Dimensionality

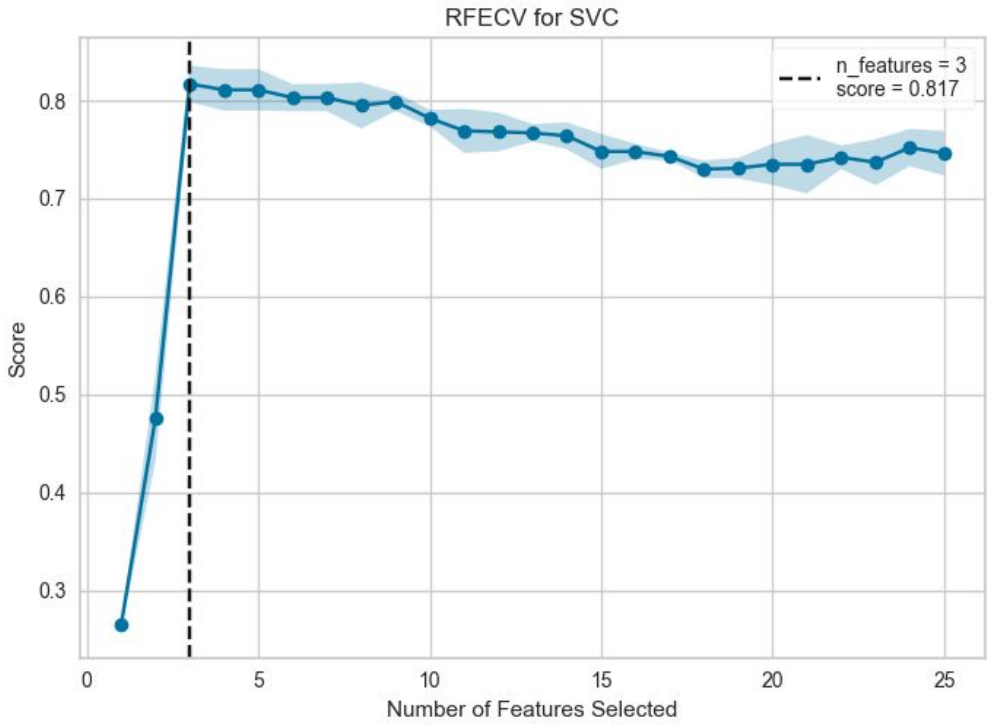
The Curse of Dimensionality

- High dimensional data tends to be sparse and far apart.
- As dimensions increase, training examples don't. This means fewer and fewer examples per region, more and more regions where you can't make any inferences.
- Algorithms based in locality need to determine nearness, but increased or decreased dimensions can skew results.



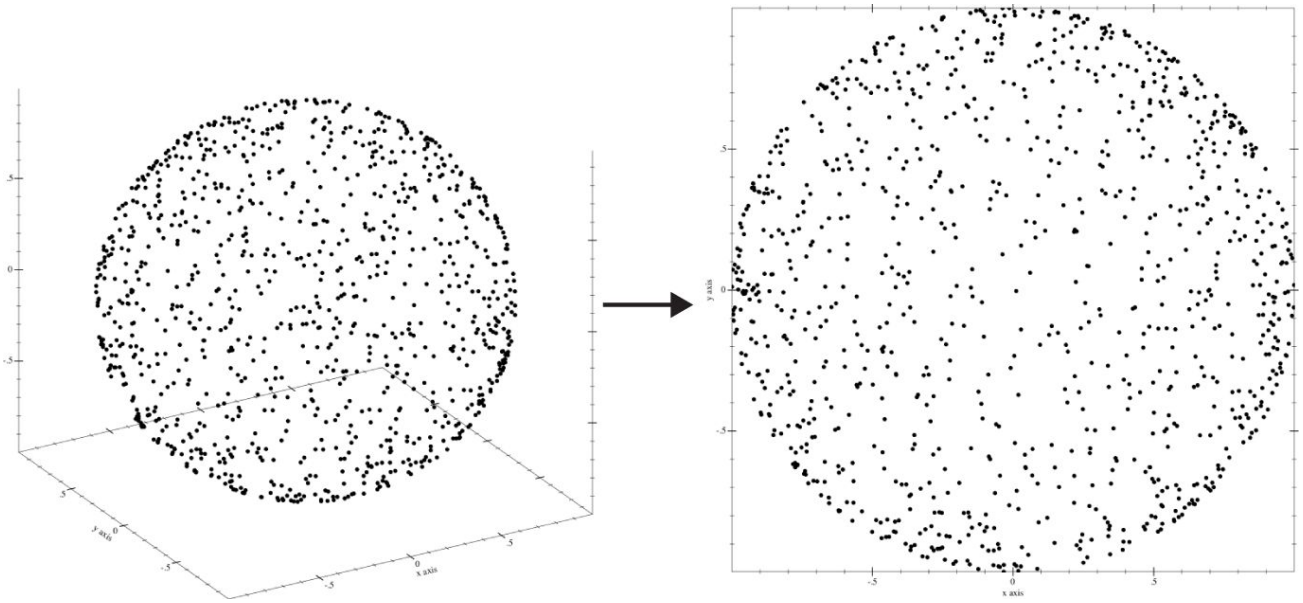
Feature Selection

- One solution: use heuristics to select subsets of features to use.
- Recursive feature elimination fits a model and removes the weakest feature(s) until the specified number is reached.
- Problem: this doesn't always scale well (think text data!).



Dimensionality Reduction

- Another solution: reduce dimensionality by projecting the original points onto a lower dimensional space.
- Here, moving from 3D to 2D reduces the average distance between points.



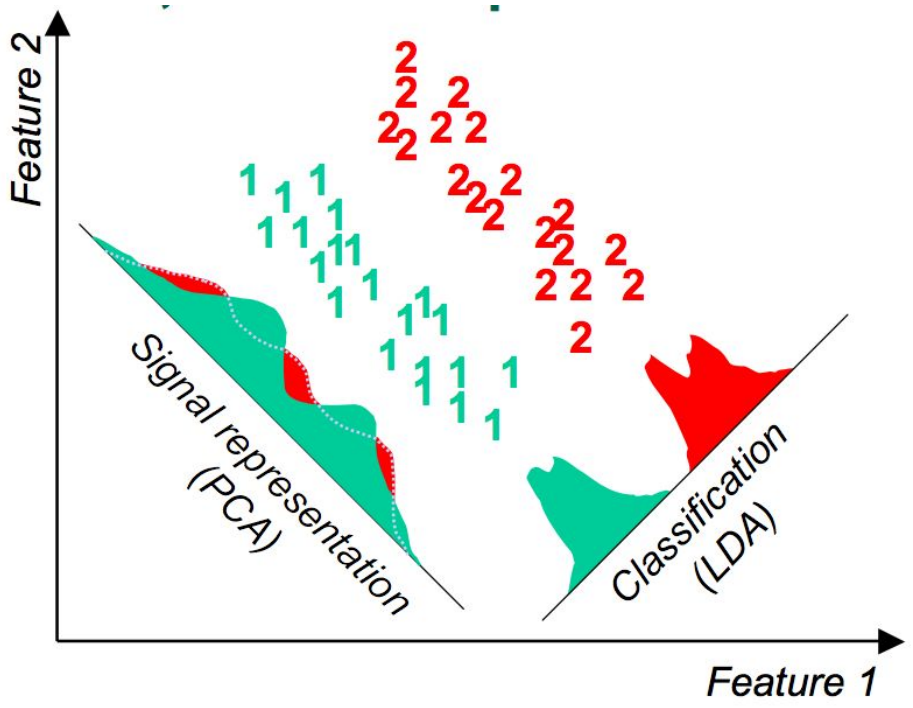
Dimensionality Reduction

Principal Components Analysis (PCA)

- An *unsupervised* method
- Uses a signal representation criterion
- Identifies combination of attributes that accounts for most variance in data.

Linear Discriminant Analysis (LDA)

- A *supervised* method
- Uses a classification criterion
- Tries to identify attributes that account for most variance between classes.



PCA

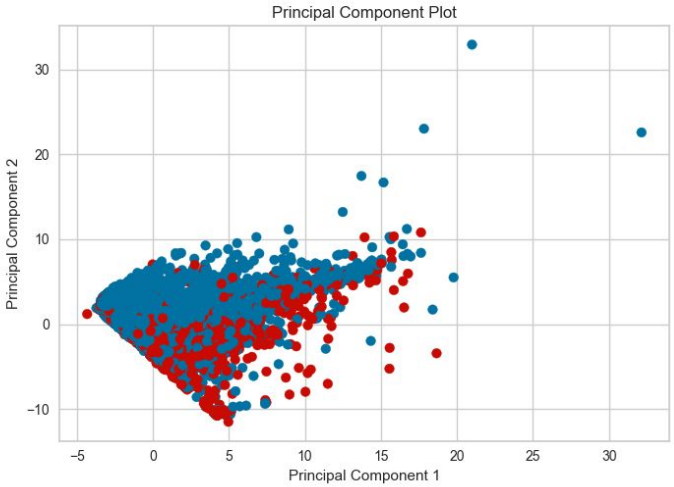
Principal Components Analysis (PCA) converts a set of observations into a set of values of **linearly uncorrelated variables** called principal components.
Scikit-learn uses a Singular Value Decomposition (SVD) of the data, keeping only the most significant singular vectors to project it into a lower dimensional space.

```
from sklearn.decomposition import PCA


X = data.features


pca = PCA(n_components=2)
new_features = pca.fit(X).transform(X)
print(new_features)
```

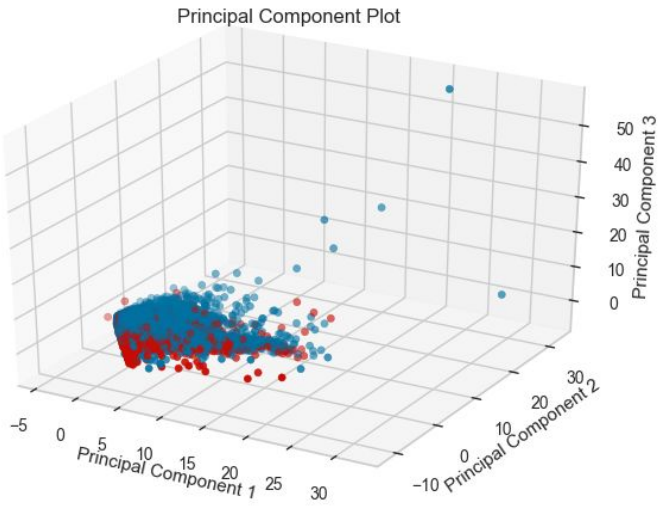
PCA Decomposition

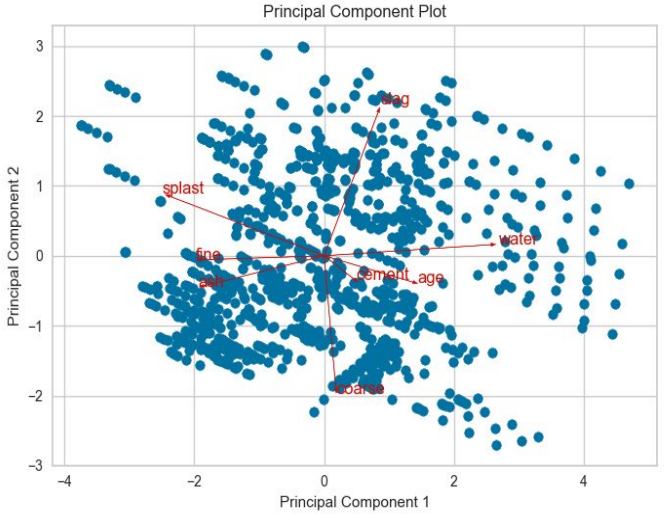
Decompose high-dimensional data into 2 or 3D, plot instances in a scatterplot.

Analyze projected data along axes of principal variation.



Can spherical distance metrics be used?





In Scikit-Yellowbrick

```
from yellowbrick.features.pca import PCADecomposition

# 2D
visualizer = PCADecomposition(scale=True, color=colors)
visualizer.fit_transform(X, y)
visualizer.poof()

# 3D
visualizer = PCADecomposition(scale=True, color=colors,
proj_dim=3)
visualizer.fit_transform(X, y)
visualizer.poof()
```

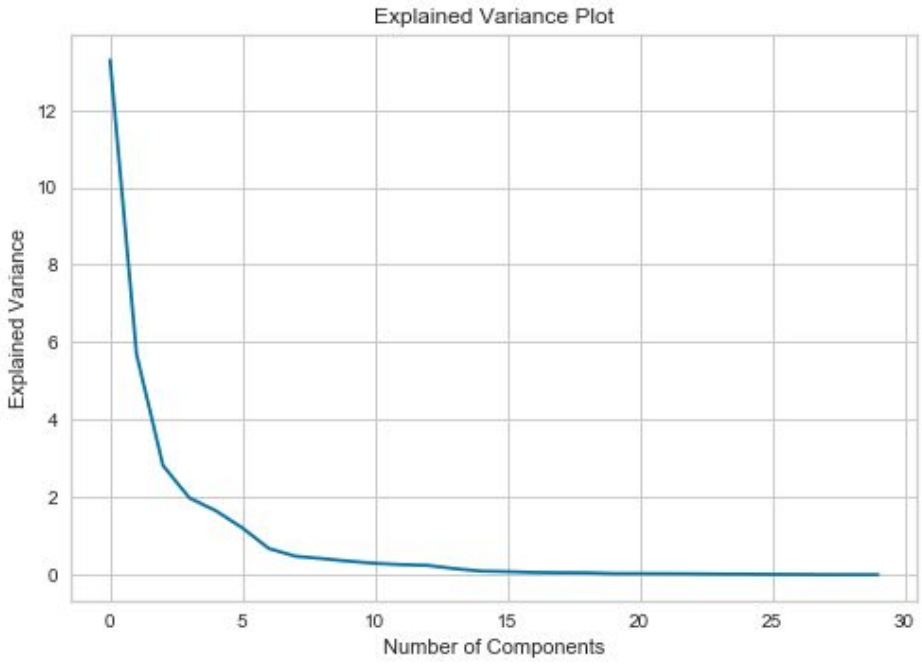
Sketch of PCA Algorithm

1. Standardize $X \rightarrow Z$ with 0 mean and unit variance.
2. Compute covariance matrix, Z^TZ
3. Calculate eigenvectors and their corresponding eigenvalues for Z^TZ via eigendecomposition into PDP^{-1}, where P is the matrix of eigenvectors and D is the diagonal matrix with eigenvalues on the diagonal and values of zero everywhere else.
4. Sort eigenvalues in P from largest to smallest into P*
5. $Z^* = ZP^* \rightarrow$ transformed data whose features are orthogonal to each other (independent).
6. Select features to keep.

Choosing Components

1. Arbitrary selection (e.g. we need 2 or 3 for plotting) or there is some other reason for a good guess at the number of features.
2. Compute the explained variance of each component and set a threshold, e.g. 80%
3. Use the Elbow method as shown to the right.

Explained variance: each eigenvalue corresponds to the importance of that component. The explained variance is therefore the ratio of the sum of kept eigenvalues to sum of all eigenvalues.



LDA: Supervised Dimensionality Reduction

A classifier with a linear decision boundary generated by fitting class conditional densities to the data and using Bayes' rule. Fits a Gaussian density to each class, assuming all classes share the same covariance matrix.

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

X = data.features
y = data.target

lda = LinearDiscriminantAnalysis(n_components=2)
new_features = lda.fit(X, y).transform(X)
print(new_features)
```

Other Linear Dimensionality Reduction

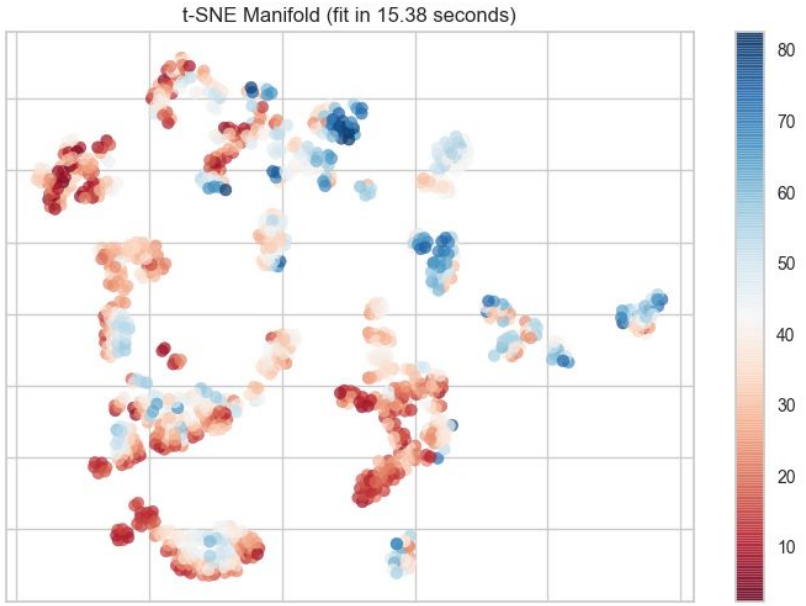
- **Incremental PCA:** if the dataset does not fit into memory, uses partial fits to decompose to a final fit.
- **SVD/TruncatedSVD/LSA**: works on sample matrices directly rather than the covariance matrix - better for sparse data (e.g. text).
- **Kernel PCA:** uses kernel method for non-linear dimensionality reduction.
- Others: Factor Analysis, Sparse Coding, ICA, NNMF

Manifolds

Manifold learning is an approach to non-linear dimensionality reduction. Algorithms for this task are based on the idea that the dimensionality of many data sets is only artificially high.

- MDS
- t-SNE
- Isomaps
- LLE
- Spectral Embedding

Precursor to Neural Manifolds and Embeddings.

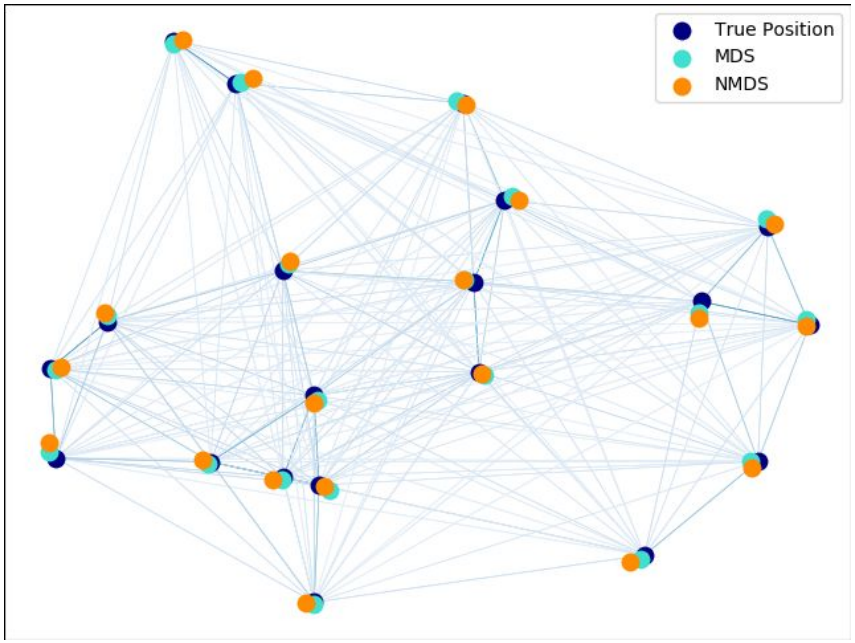


Multi-Dimensional Scaling

Similarity == Distance:

Metric MDS: create similarity matrix from a distance metric and decompose so that output points are as close to the similarity data as possible.

Non-Metric MDS: preserve the order of the distances such that there is a monotonic relationship between the distance in embedded space and the original space.



What is an Anomaly?

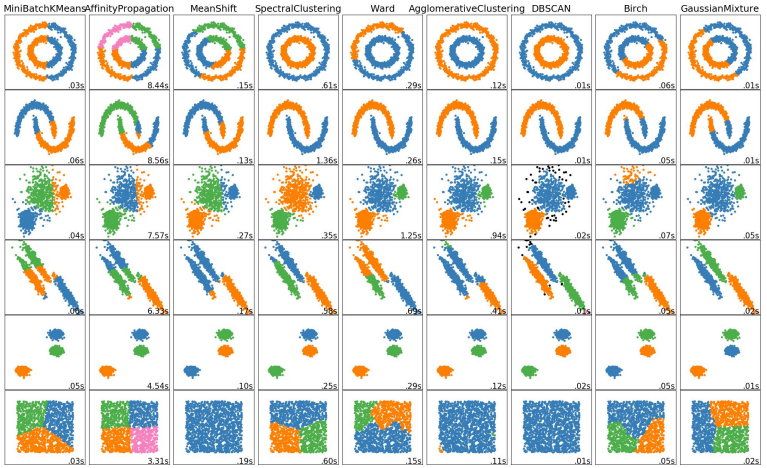
Anomaly Detection

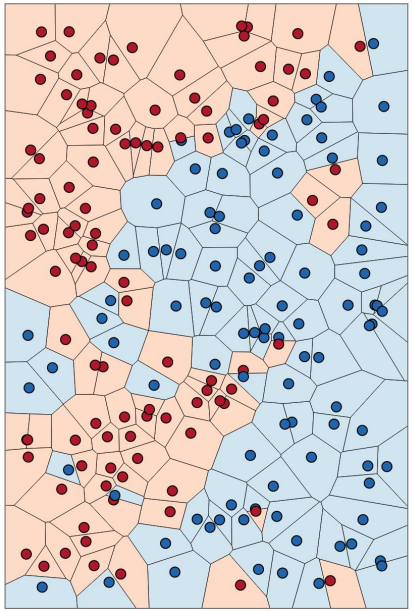
Identifying instances that are different from the majority of the data:

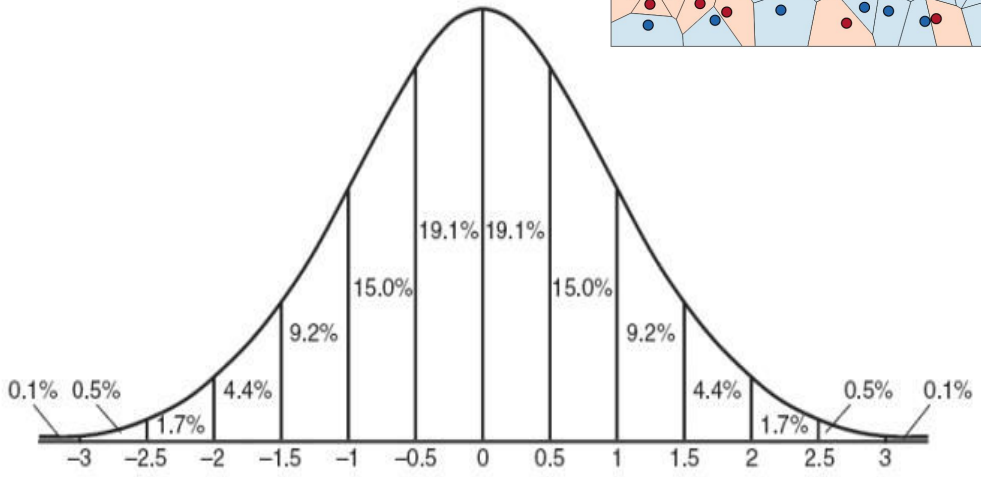
- Errors
- Exotics
- Rare events
- Emergent trends
- Equipment failures
- Fraudulent activities

Intuition

We should be able to combine what we've learned in this class (e.g. classification & clustering) with what know about outliers (e.g. z-scores) to identify anomalies in high-dimensional data, using both parametric and non-parametric (e.g. proximity-based) models.







Distinction: Outliers vs. Novelties

Detecting outliers

- Training data *does* contains outliers (observations far from the others).
- Want to try to fit the regions where the training data is the most concentrated, *ignoring the deviant observations*.

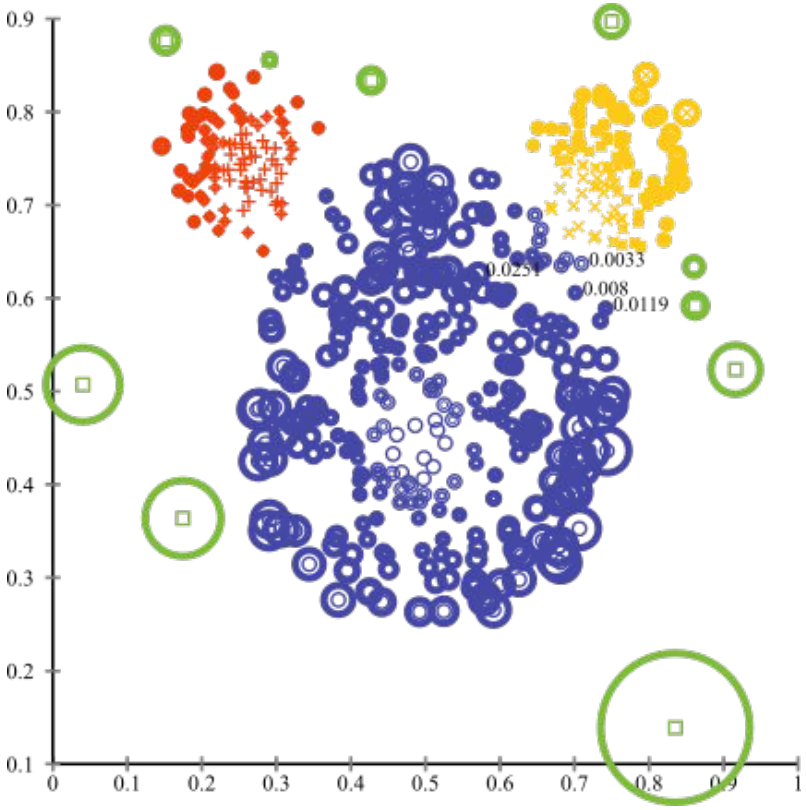
Detecting novelties

- Training data *does not* contain outliers.
- Want to detect whether a *new* observation is an outlier (aka a novelty).

Unsupervised Techniques for Anomaly Detection

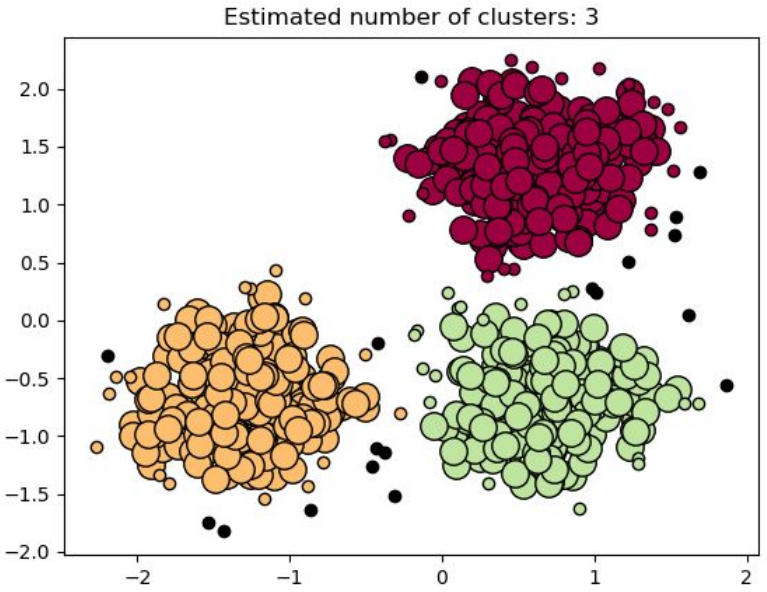
Clustering Approach

- Use clustering to find high-density grouping within the data
- Identify individual instances that are *outliers* -- points that fall outside natural grouping patterns.
- Inspect points in very small clusters for candidate outliers.
- Compute the distance between candidate outliers and high density clusters.



DBSCAN

- “Density-Based Spatial Clustering of Applications with Noise”
- Finds core samples of high density and expands clusters from them.
- Unlike KMeans, don’t need to know number of clusters in advance.
- Good for data which contains clusters of similar density.



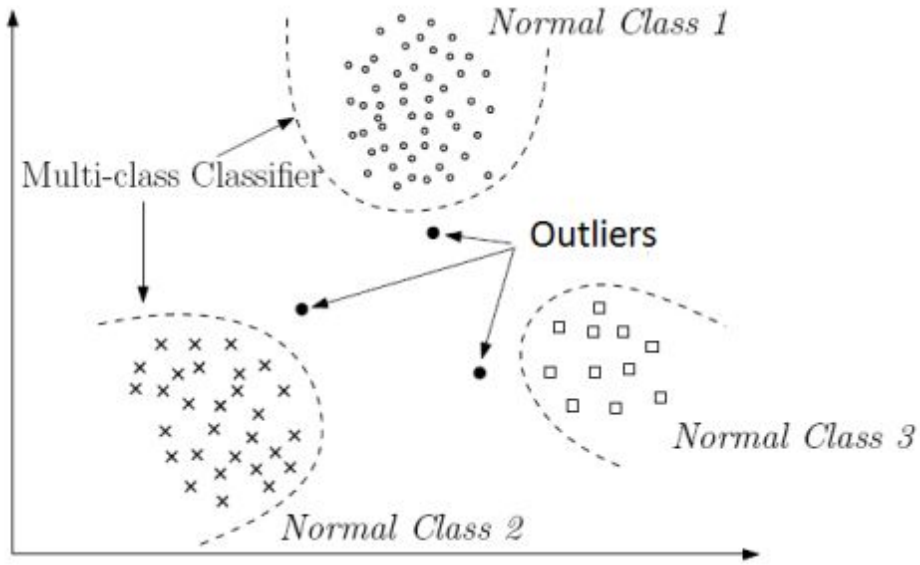
In Scikit-Learn

```
import numpy as np
from sklearn.cluster import DBSCAN


X = np.array([[1, 2], [2, 2], [2, 3],
              [8, 7], [8, 8], [25, 80]])
max_dist = 3 # max distance between 2 samples to be a cluster
min_samp = 2 # smallest size of a cluster

cluster_scan = DBSCAN(eps=max_dist, min_samples=min_samp).fit(X)
print(cluster_scan.labels_) # outliers will be labeled -1
```

Semi-Supervised Techniques for Anomaly Detection

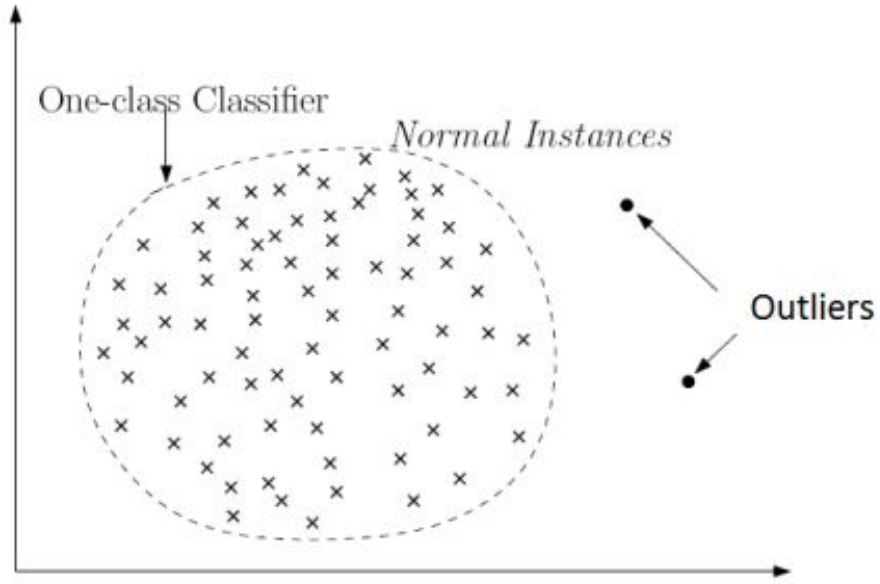
Classification Approach: Multi-class

- Construct a multi-class classifier from data without outliers.
- Use the trained classifier on new data that may contain outliers.
- Use prediction probabilities to estimate the likelihood that a given point is an outlier.



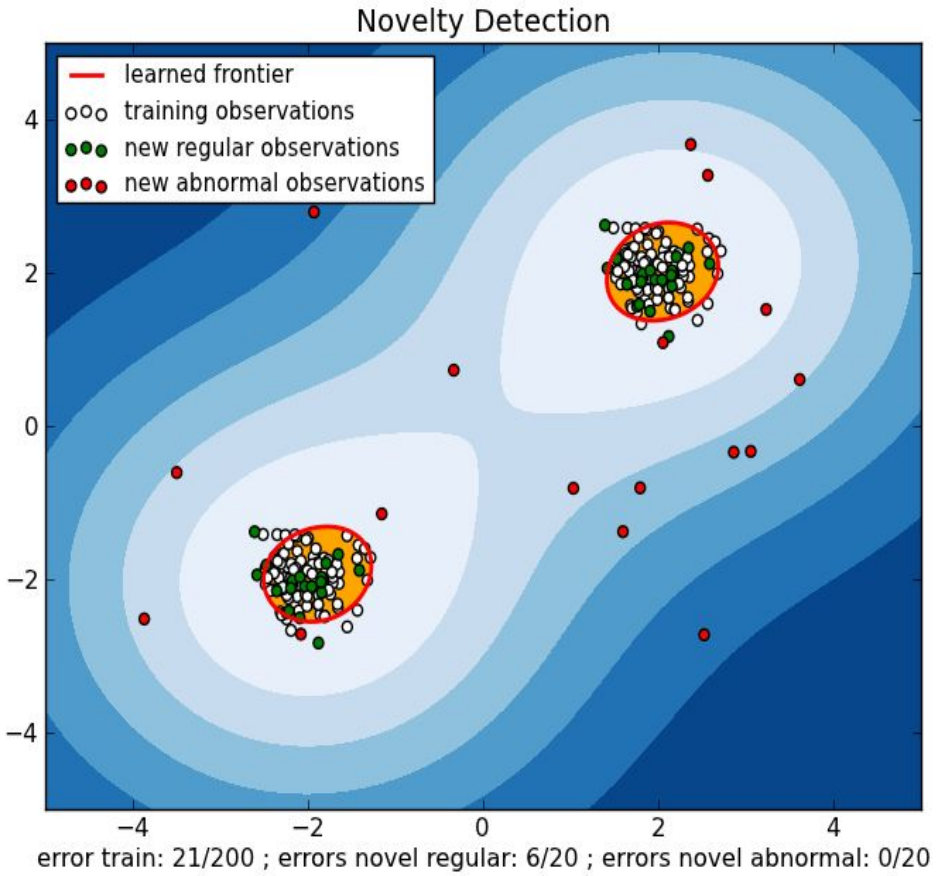
Classification Approach: One-class

- Assume all training instances belong to a single class.
- Train a one-class estimator to learn the boundary around the normal instances.
- Any test instance that falls outside the learned boundary is an outlier.



One Class SVM

- An unsupervised algorithm that learns a decision function.
- Estimates the support of a high-dimensional distribution.
- Classifies new data as similar to or different from the training set.
- Best suited for novelty detection (training set contains no outliers) or for anomaly detection on very high dimensional data.

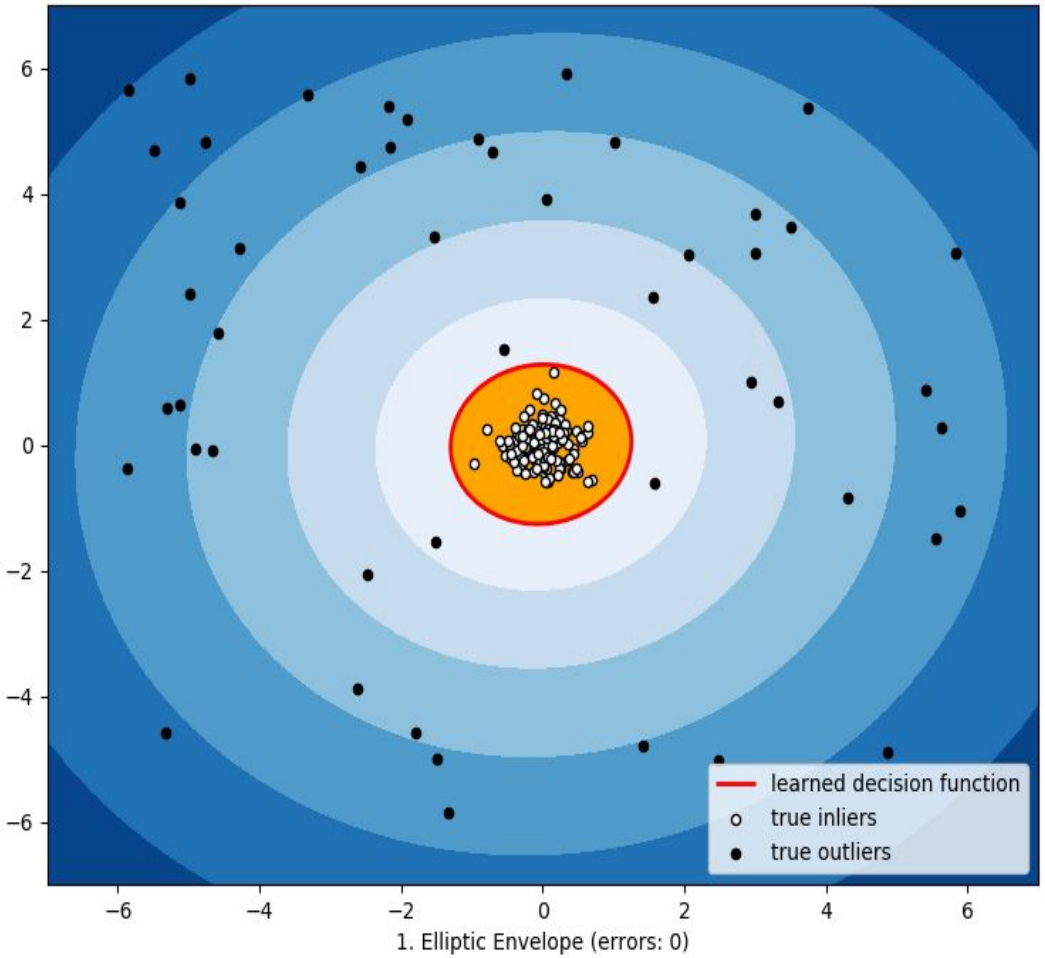


In Scikit-Learn

```
from sklearn.svm import OneClassSVM
from sklearn.preprocessing import Normalizer


X = Normalizer().fit_transform(data.features) # normalize first
nu = outliers / target # proportion of data that is outliers
kernel = ‘rbf’ # similarity function over pairs of points
gamma = 0.00005 # influence of individual samples (smoothness)

svm = OneClassSVM(nu=nu, kernel=kernel, gamma=gamma).fit(X)
svm.predict(test_pts) # returns +1 for normal, -1 for novel
```

Elliptic Envelope

- Assumes data is Gaussian and fits the smallest volume ellipsoid.
- Discards specified fraction of contamination points, so robust to outliers.
- Degrades with high-dimensional data (make sure `n_samples > n_features ** 2`).
- Best suited to large data sets.

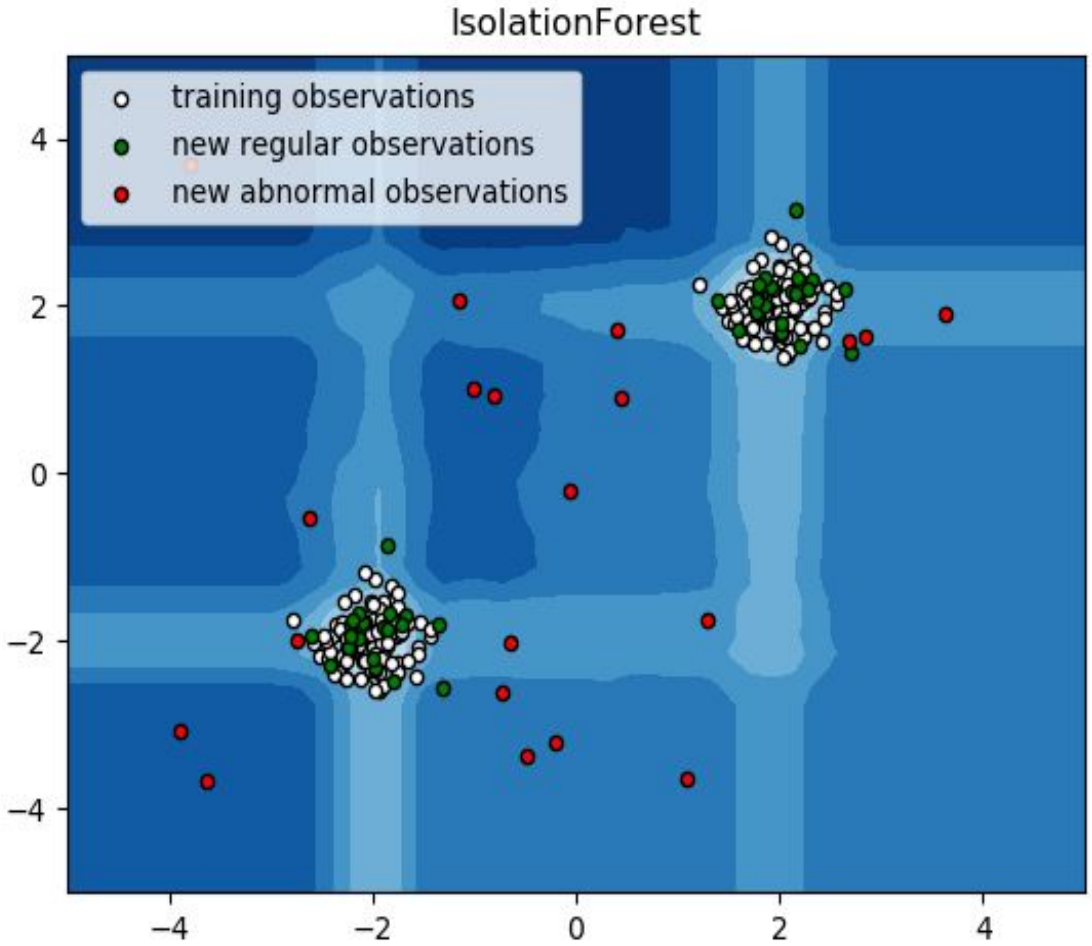


In Scikit-Learn

```
from sklearn.covariance import EllipticEnvelope


X = data.features
outlier_frac = 0.035 # fraction of points estimator can discard

ellipse = EllipticEnvelope(contamination=outlier_frac).fit(X)
ellipse.predict(test_pts) # outliers are labeled -1
```

Isolation Forests

- Selects features and randomly splits values for that feature between its max and min.
- # of splits required to isolate a sample equals path length from root to terminating node.
- Path length is a measure of normality; a shorter path for a sample indicates a likely anomaly.
- Well suited to high dimensional data.



In Scikit-Learn

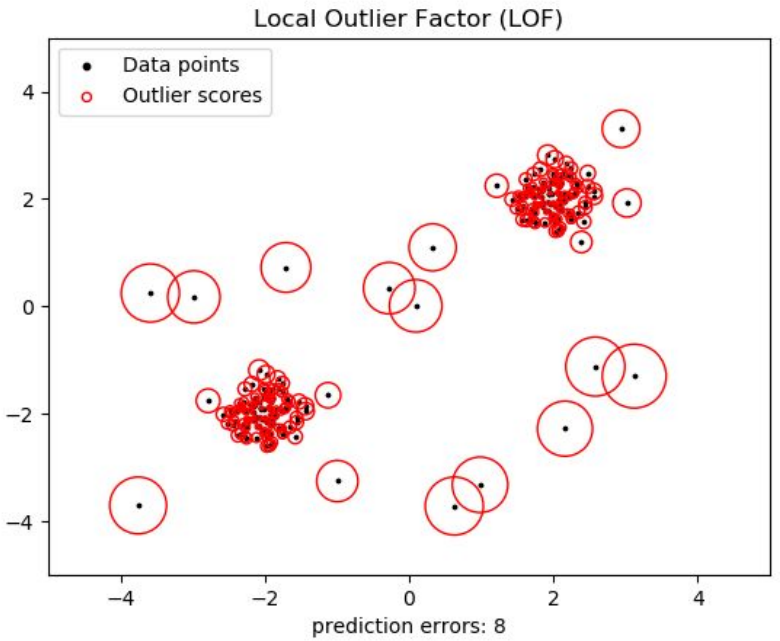
```
from sklearn.ensemble import IsolationForest


X = data.features
max = 10 # num features from X to use to train each forest
outlier_frac = 0.015 # fraction of points estimator can discard

forest = IsolationForest(
    max_features=max, contamination=outlier_frac
).fit(X)
forest.predict(test_pts) # outliers are labeled -1
```

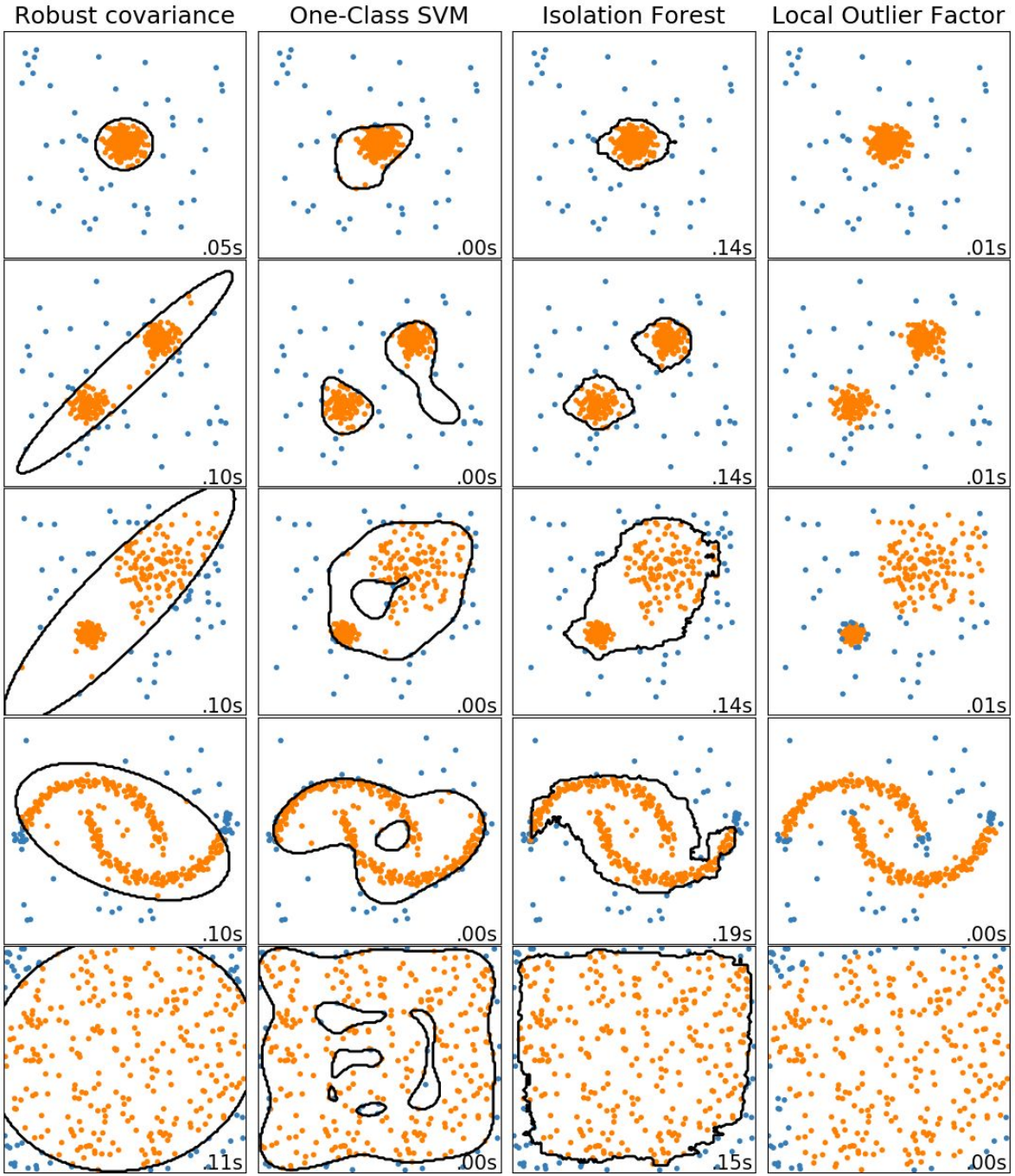
Local Outlier Factor

Unsupervised method for outlier detection in an instance-based fashion.

Each training point is assigned an **anomaly score:** the derivation of the density of the point with respect to its neighbors.

Locality is given by k-nearest-neighbors points with a substantially lower density then its neighbors are considered outliers.




Robust covariance
One-Class SVM
Isolation Forest
Local Outlier Factor
.05s
.00s
.14s
.01s
.10s
.00s
.14s
.01s
.10s
.00s
.14s
.01s
.10s
.00s
.19s
.00s
.11s
.00s
.15s
.00s

Case Study: Bot Detection

Can we differentiate bots from people?

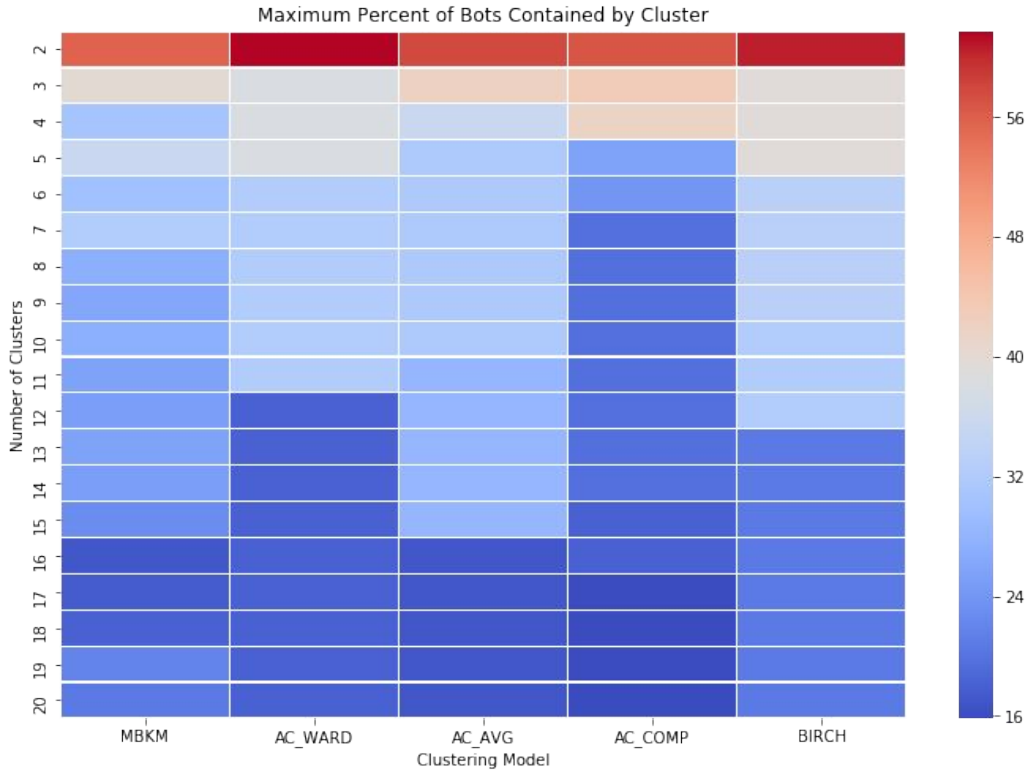
- Capstone Project by Joel Alejandre, Silva Kurtisa, Alexis Starr (Dec 2017)
- Browser fingerprints contain information about remote computing devices, e.g.:
  - browser type
  - fontlist
  - screen resolution
  - addblocker enabled
  - JS enabled
  - useragent string
- Can these features be used to distinguish human traffic from bots?

Problem: How to Validate?

- Bad bots don’t come out and confess to being bots! In other words, no labeled data.
- How do you validate if you’ve successfully captured bots through your anomaly detection techniques?
- Good bots don’t mask that they are bots; their fingerprints contain flags that acknowledge that they are bots.
- Hypothesis: Bots are bots, if we can build a model that effectively classifies the good bots, we can use it to look for likely bad bots.

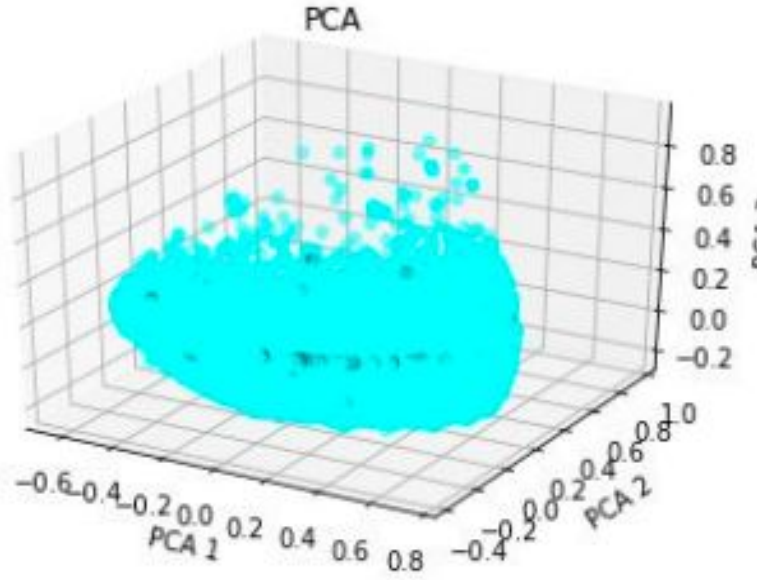
Clustering fingerprints

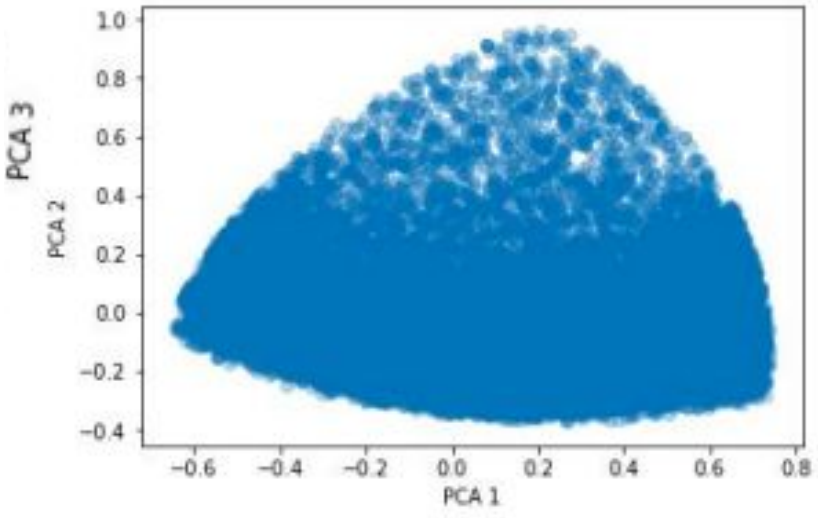
- Agglomerative and Birch clustering successfully identified the highest percentage of bots within a single cluster (61.75% and 60.66%, respectively).
- Increasing the number of clusters beyond n=2 did not improve accuracy of the models.



PCA

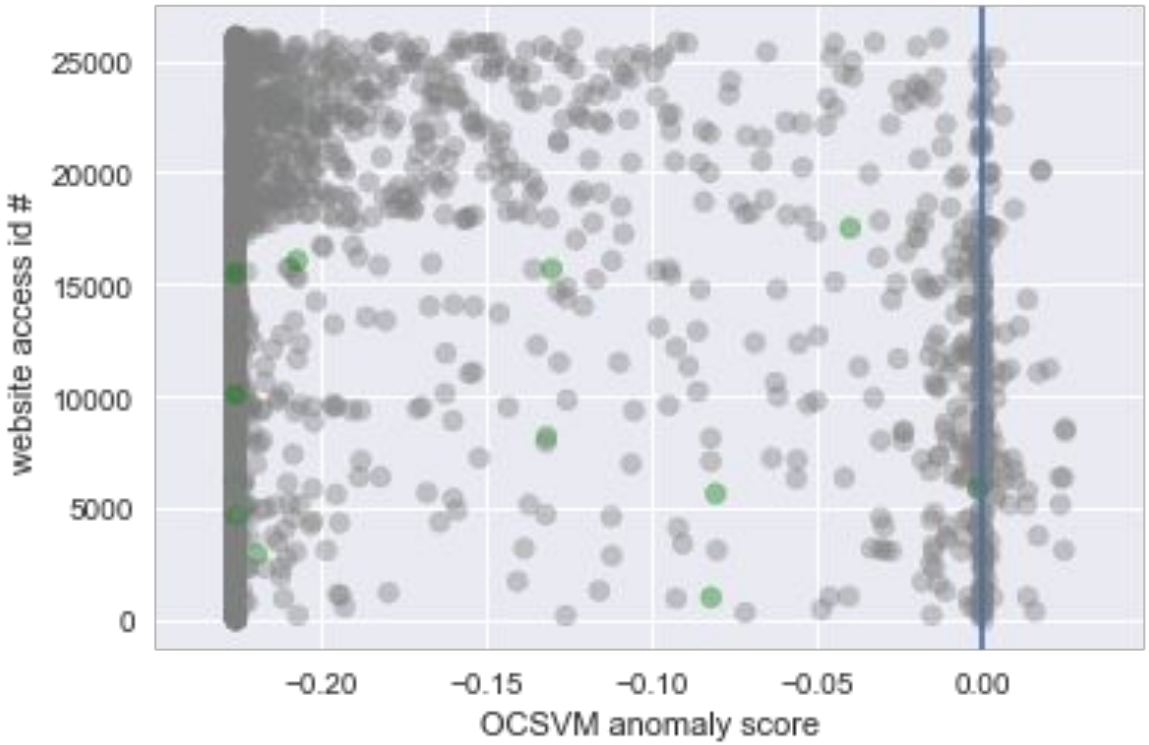
- Principal component analysis with 3 components
- No clear separation between “good bots” and the other web browser instances
- Not successful in identifying the bot profiles





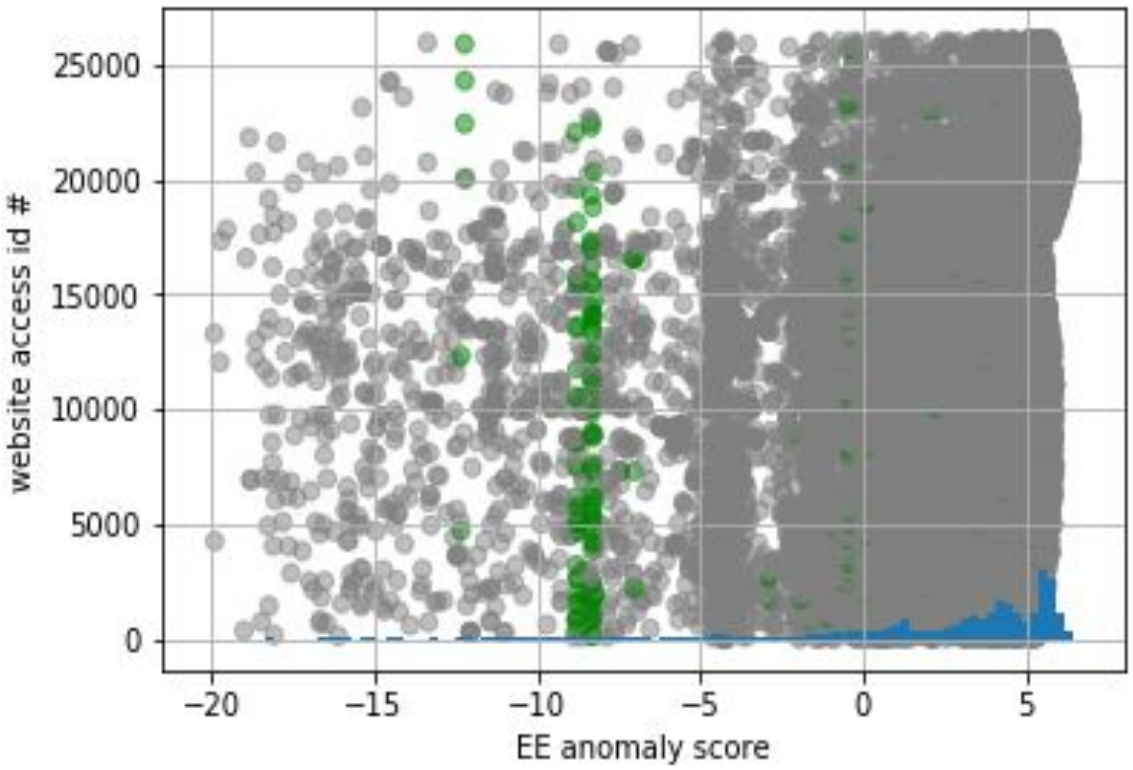
One Class SVM

GoodBot	0	1	All
SVM			
-1	12023	61	12084
1	13904	122	14026
All	25927	183	26110



Elliptic Envelope

GoodBot	0	1	All
EE			
-1	3773	144	3917
1	22154	39	22193
All	25927	183	26110



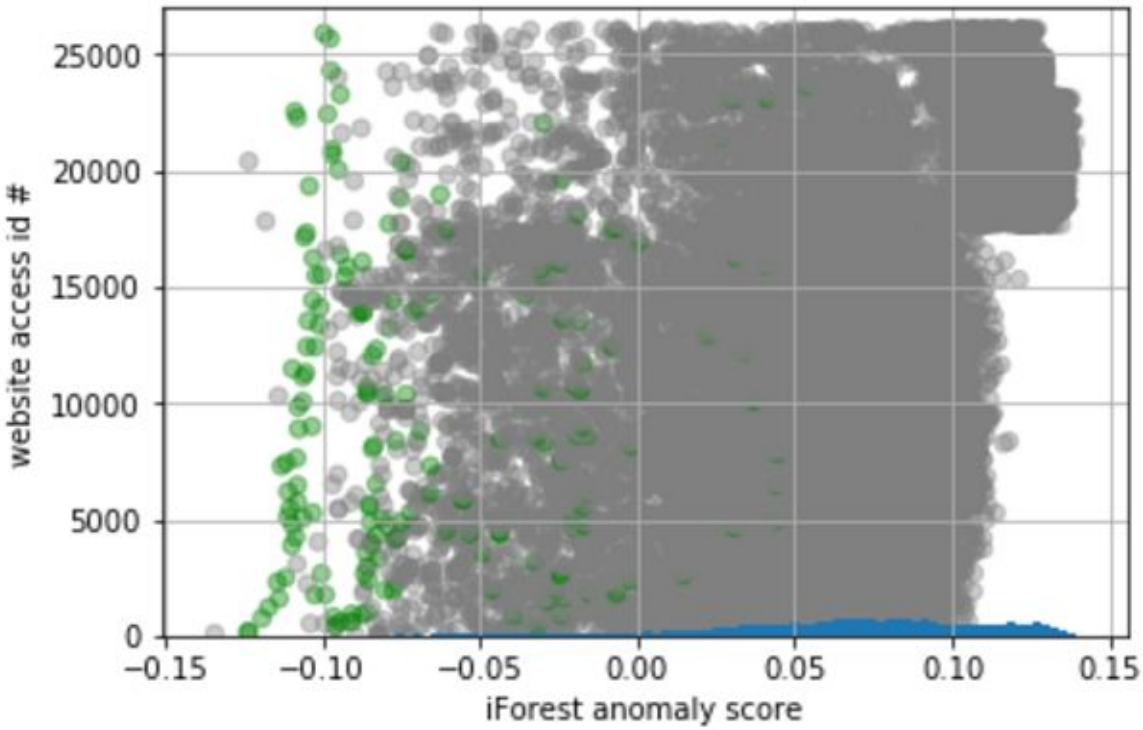
Isolation Forest

GoodBot IFT	0	1	All
-1	417	20	437
1	2690	7	2697
All	3107	27	3134

74.1%

GoodBot IFT	0	1	All	
-1	3765	152	3917	
1	22162		31	22193
All	25927		183	26110

83.1%



Anomaly Detection Results Summary

- PCA: unable to separate “good bots” from humans.
- `OneClassSVM`: incorrectly classified “good bots” as human.
- `IsolationForest`: classified 74% of “good bots” as anomalies.
- `EllipticEnvelope`: classified 82% of “good bots” as anomalies.