KNN for outlier detection

ANOMALY DETECTION IN PYTHON

Bekhruz (Bex) Tuychiev

Kaggle Master, Data Science Content Creator

Applications of KNN

- Supervised:
  - Regression
  - Classification
- Unsupervised:
  - Clustering
  - Outlier detection

Simplicity of KNN

Anomaly scores are calculated in

- Isolation Forest:
  - Tree depth
  - Sub-sample size
  - Many other components
- KNN:
  - Only the distance between instances

Ansur Male Dataset

```
import pandas as pd

males = pd.read_csv("ansur_male.csv")
males.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4082 entries, 0 to 4081
Data columns (total 95 columns):
 #   Column                          Non-Null Count  Dtype
 0   abdominalextensiondepthsitting  4082 non-null   int64
 1   acromialheight                  4082 non-null   int64
 2   acromionradialelength           4082 non-null   int64
 3   anklecircumference              4082 non-null   int64
 4   axillaheight                    4082 non-null   int64
  ...
```

KNN in action

```
from pyod.models.knn import KNN

knn = KNN(contamination=0.01, n_jobs=-1)

knn.fit(males)
```

KNN with outlier probabilities

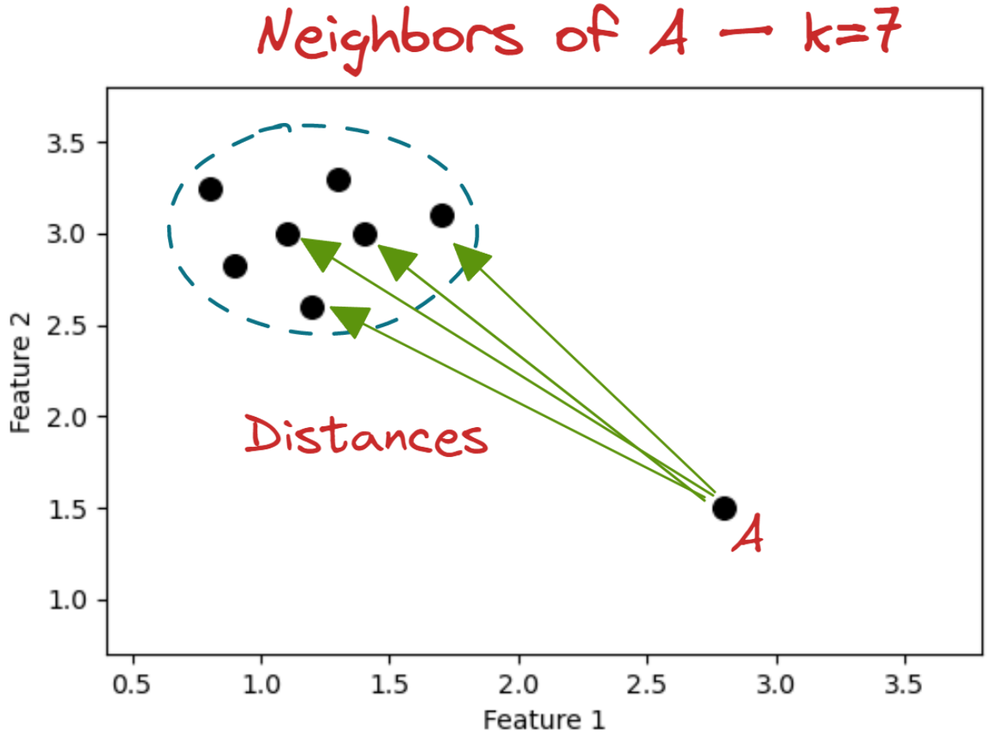
```
probs = knn.predict_proba(males)

# Use 55% threshold for filtering
is_outlier = probs[:, 1] > 0.55

# Isolate the outliers
outliers = males[is_outlier]

len(outliers)
```

```
13
```

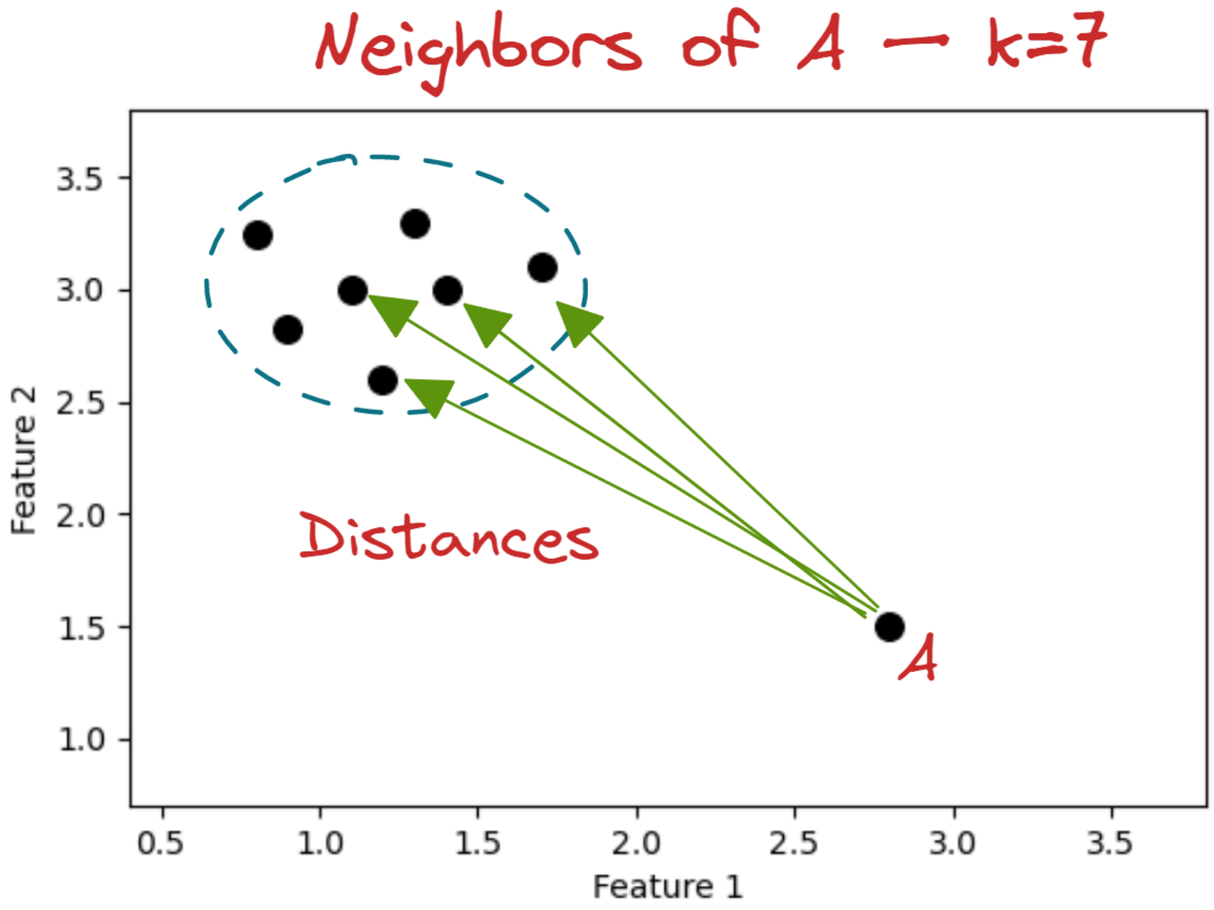
The number of neighbors

```
# k=20 when contamination is <=10%
knn = KNN(n_neighbors=20, n_jobs=-1)
knn.fit(males)

probs = knn.predict_proba(males)

is_outlier = probs[:, 1] > .55
outliers = males[is_outlier]

len(outliers)
```

```
15
```



Features of KNN



Drawbacks of KNN

- Memorizes the dataset - memory-inefficient
- Slow prediction stage
- Sensitive to feature scales

Let's practice!

ANOMALY DETECTION IN PYTHON

Outlier-robust feature scaling

ANOMALY DETECTION IN PYTHON

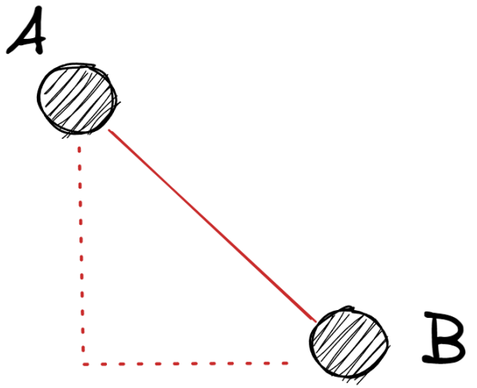
Bekhruz (Bex) Tuychiev

Kaggle Master, Data Science Content Creator

Euclidean distance

```
A = np.array([9, 1, 6])
B = np.array([25, 44, 85])

diffs = (B - A) ** 2
dist_AB = np.sqrt(np.sum(diffs))

print(dist_AB)
```

```
91.35644476444998
```



$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space

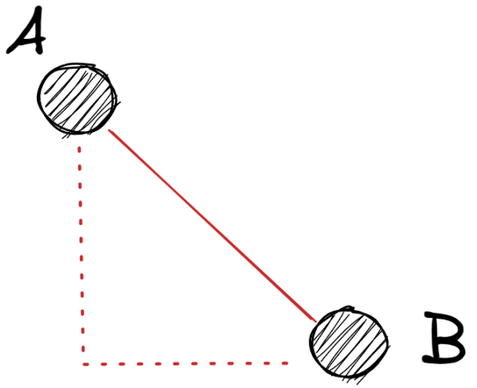
q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

Euclidean in SciPy

```
from scipy.spatial.distance import \
euclidean

dist_AB = euclidean(A, B)
dist_AB
```

```
91.35644476444998
```



$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space

q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

Standardization

- Mean is subtracted and divided by the STD
- Result: mean of zero and STD of 1

StandardScaler

```
from sklearn.preprocessing import StandardScaler


ss = StandardScaler()


# Extract feature and target
X = males.drop("weightkg", axis=1)
y = males[['weightkg']]


# Fit
ss.fit(X)
```

Transforming

```
X_transformed = ss.transform(X)


X_transformed[:5]
```

```
array([[-1.05174523],
       [-0.29289108],
       [ 1.3446363 ],
       [-1.21654894],
       [0.056451235]])
```

fit_transform

```
ss = StandardScaler()


X_transformed = ss.fit_transform(X)
```

QuantileTransformer

```
from sklearn.preprocessing import QuantileTransformer


# Init
qt = QuantileTransformer()
X = males.drop("weightkg", axis=1)
y = males[['weightkg']]

X_transformed = qt.fit_transform(X)

X_transformed.shape
```

```
(4082, 94)
```

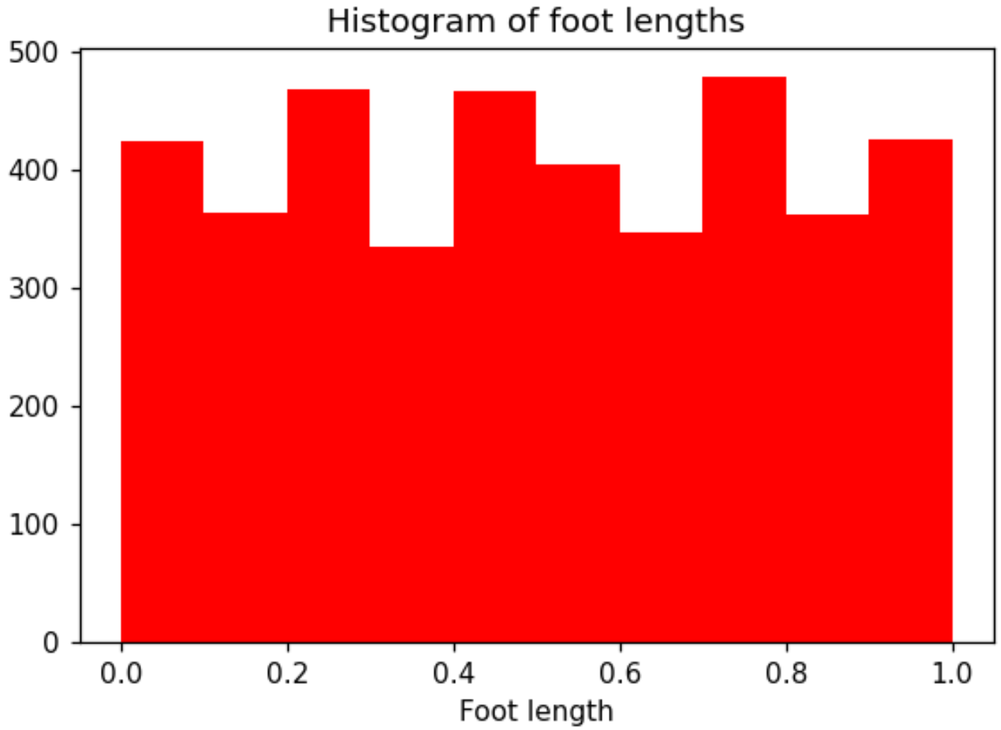
Preserving column names

```
qt = QuantileTransformer()

X.loc[:, :] = qt.fit_transform(X)

X.head()
```

	abdominalextensiondepthsitting	acromialheight	acromionradialelength	anklecircumference	axillaheight	balloffootcircumference	balloffootlength
0	0.556557	0.174675	0.263764	0.220721	0.139139	0.310811	0.310811
1	0.118619	0.778278	0.696697	0.285786	0.828829	0.405405	0.350350
2	0.046046	0.724224	0.852352	0.875375	0.720220	0.779780	0.920921
3	0.832332	0.652653	0.385886	0.480480	0.592092	0.849850	0.733233
4	0.988989	0.281782	0.440440	0.480480	0.300300	0.173173	0.506006

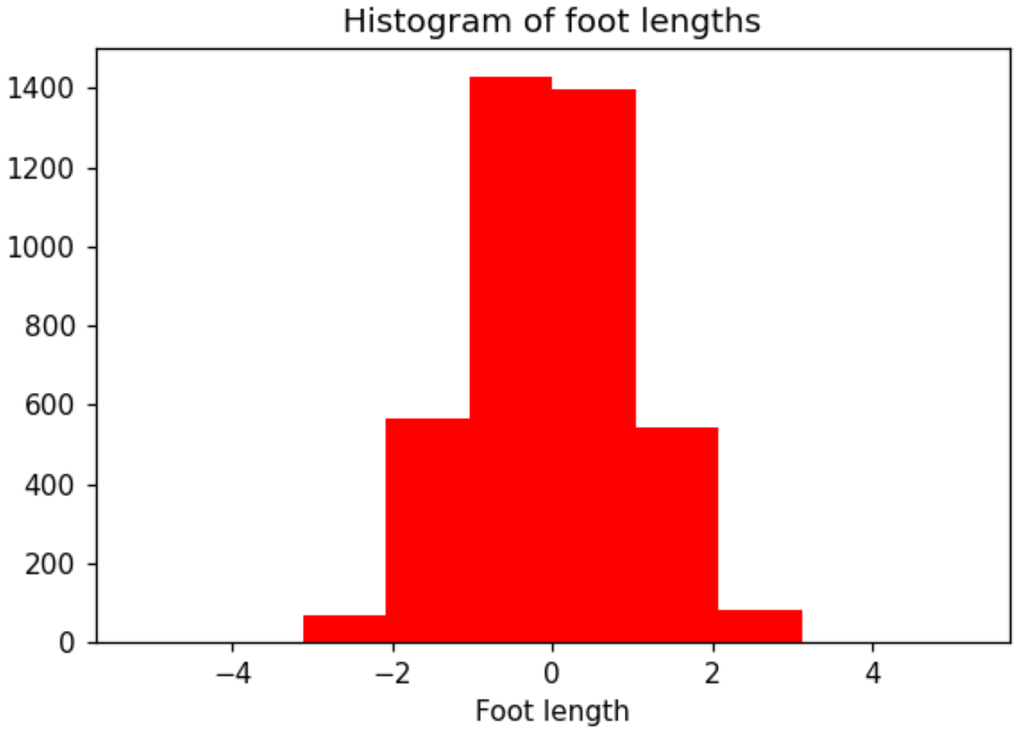
Uniform histogram

```
plt.hist(X['footlength'], color='red')

plt.xlabel("Foot length")
plt.title("Histogram of foot lengths")

plt.show()
```



Normal histogram

```
qt = QuantileTransformer(
  output_distribution='normal')

# Rebuild the overridden feature array
X = males.drop("weightkg", axis=1)
X.loc[:, :] = qt.fit_transform(X)

plt.hist(X['footlength'], color='r')
plt.xlabel("Foot length")
plt.title("Histogram of foot lengths")

plt.show()
```



Let's practice!

ANOMALY DETECTION IN PYTHON

Hyperparameters of KNN

ANOMALY DETECTION IN PYTHON

Bekhruz (Bex) Tuychiev

Kaggle Master, Data Science Content Creator

datacamp

Modify evaluate_outlier_classifier

```
def evaluate_outlier_classifier(model, data, threshold=.75):
    model.fit(data)

    probs = model.predict_proba(data)
    inliers = data[probs[:, 1] <= threshold]

    return inliers
```

Modifying evaluate_regressor

```
def evaluate_regressor(inliers):
    X, y = inliers.drop("weightkg", axis=1), inliers[['weightkg']]
    X_train, X_test, y_train, y_test = train_test_split(
      X, y, random_state=10, train_size=0.8)

    lr = LinearRegression()
    lr.fit(X_train, y_train)

    preds = lr.predict(X_test)
    rmse = mean_squared_error(y_test, preds, squared=False)

    return round(rmse, 3)
```

Tuning the number of neighbors

```
n_neighbors = [5, 10, 15, 20]
scores = dict()

for k in n_neighbors:
    # Init a KNN
    knn = KNN(n_neighbors=k)

    # Get the inliers with KNN
    inliers = evaluate_outlier_classifier(knn, males_transformed, .55)

    # Calculate and store RMSE
    scores[k] = evaluate_regressor(inliers)
```

Inspecting the result

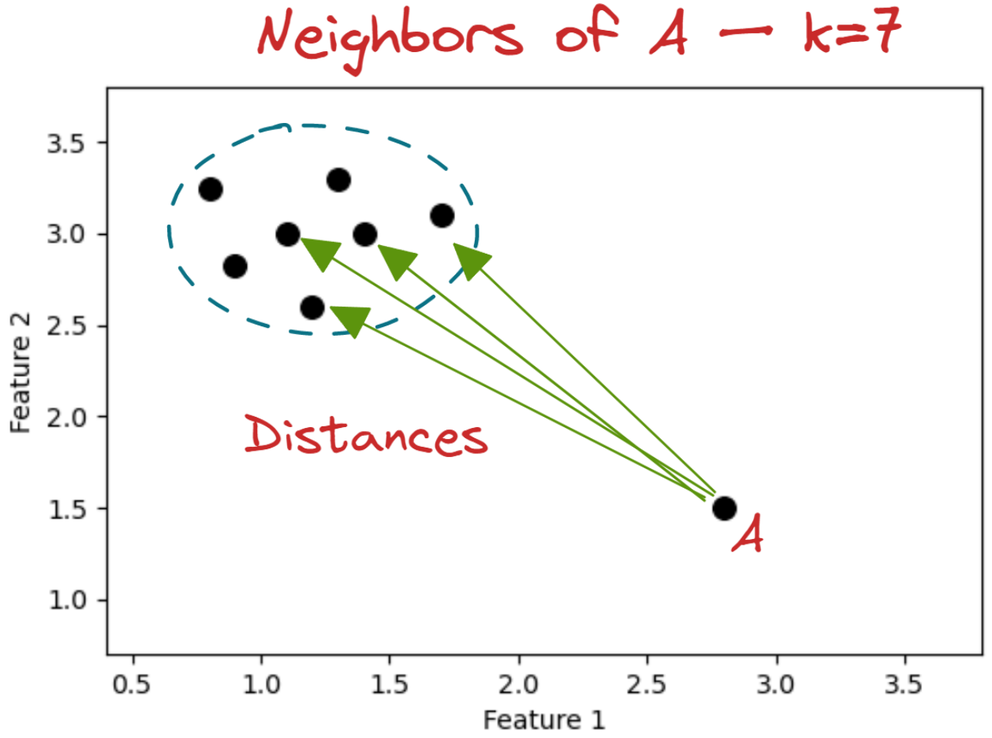
```
print(scores)
```

```
{5: 19.463, 10: 17.965, 15: 18.817, 20: 20.597}
```

Distance metrics

- More than 40 metrics are supported

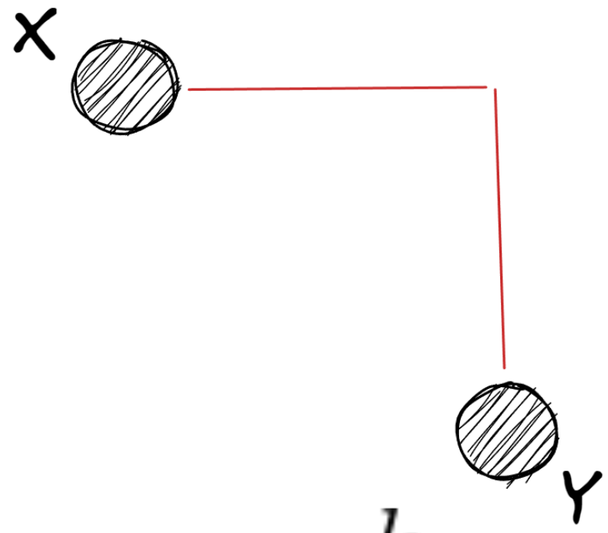
```
knn = KNN(metric='euclidean')
```



Manhattan distance

```
A = np.array([9, 1, 6, ...])
B = np.array([25, 44, 85, ...])

diffs = np.abs(B - A)

manhattan_dist_AB = np.sum(diffs)
manhattan_dist_AB
```

```
573
```

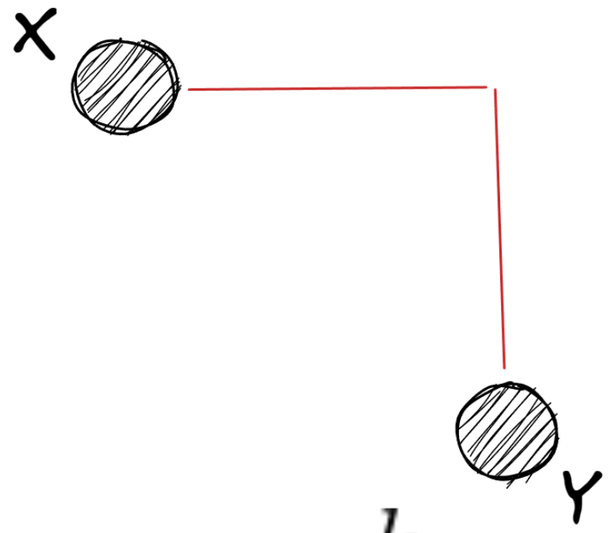


$$D(x, y) = \sum_{i=1}^{k} |x_i - y_i|$$

Manhattan distance

Manhattan distance

- Works well with high-dimensional data
- Returns larger values than euclidean
- Works best with categorical features



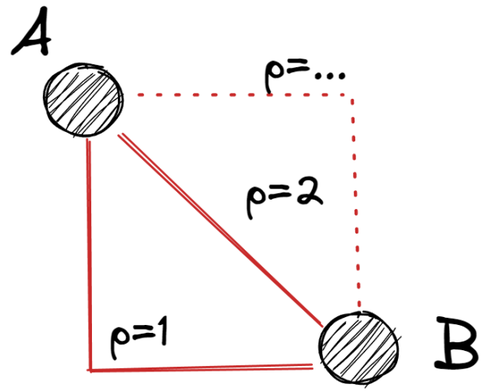
$$D(x, y) = \sum_{i=1}^{k} |x_i - y_i|$$

Manhattan distance

Minkowski distance

```
# Euclidean distance
knn = KNN(metric="minkowski", p=2)
```

```
# Manhattan distance
knn = KNN(metric="minkowski", p=1)
```



$$D(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Minkowski distance

Distance aggregation

```
knn_largest = KNN(n_neighbors=10, method="largest")


knn_mean = KNN(n_neighbors=10, method="mean")
knn_median = KNN(n_neighbors=10, method="median")
```

Tuning distance and method

```
ps = [1, 2, 3, 4]
methods = ["largest", "mean", "median"]
scores = dict()

for p, method in product(ps, methods):
    # Init an instance of KNN
    knn = KNN(n_neighbors=10, method=method, p=p, n_jobs=-1)

    # Find the inliers with the current KNN
    inliers = evaluate_outlier_classifier(knn, males_transformed, .55)

    # Calculate and store RMSE into scores
    scores[(p, method)] = evaluate_regressor(inliers)
```

Inspecting the result

```
print(scores)
```

```
{(1, 'largest'): 23.188,
 (1, 'mean'): 23.188,
 (1, 'median'): 23.188,
 (2, 'largest'): 17.965,
 (2, 'mean'): 19.463,
 (2, 'median'): 19.463,
 (3, 'largest'): 17.965,
 (3, 'mean'): 19.463,
 (3, 'median'): 19.463, ...
}
```

Let's practice!

ANOMALY DETECTION IN PYTHON

Local Outlier Factor

ANOMALY DETECTION IN PYTHON

Bekhruz (Bex) Tuychiev
Data Science Writer

datacamp

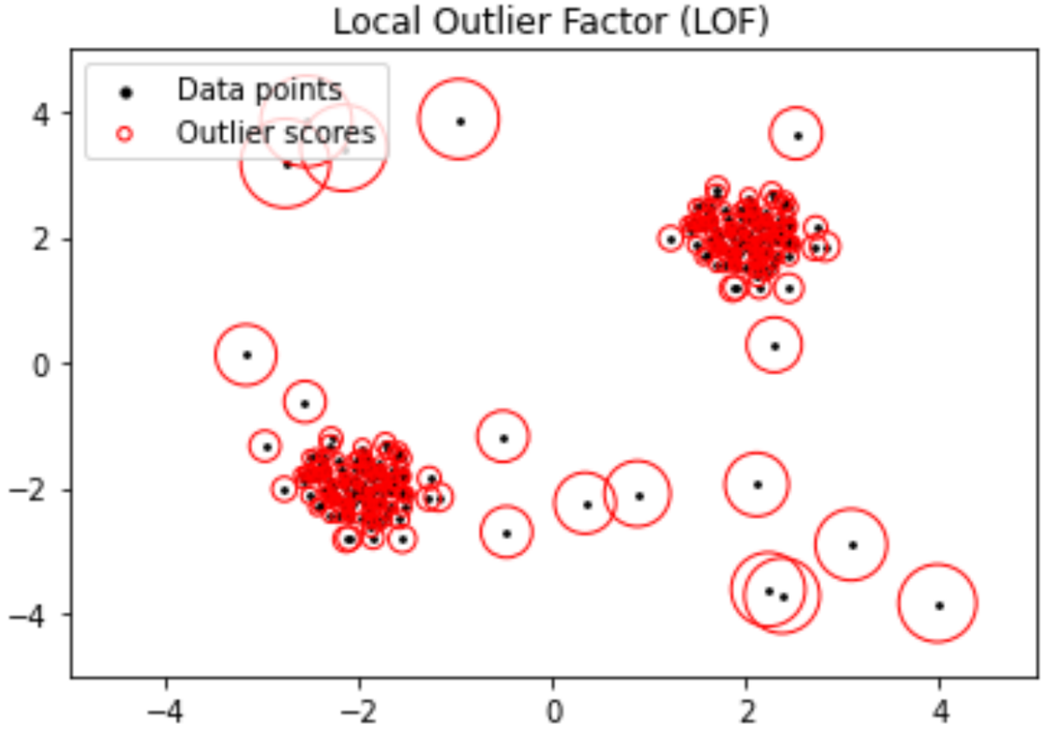
What is Local Outlier Factor (LOF)?

Features

- Density-based algorithm
- Proposed in 2000
- Works well with moderately high-dimensional data
- Can be faster than KNN and Isolation Forest

How does LOF work?

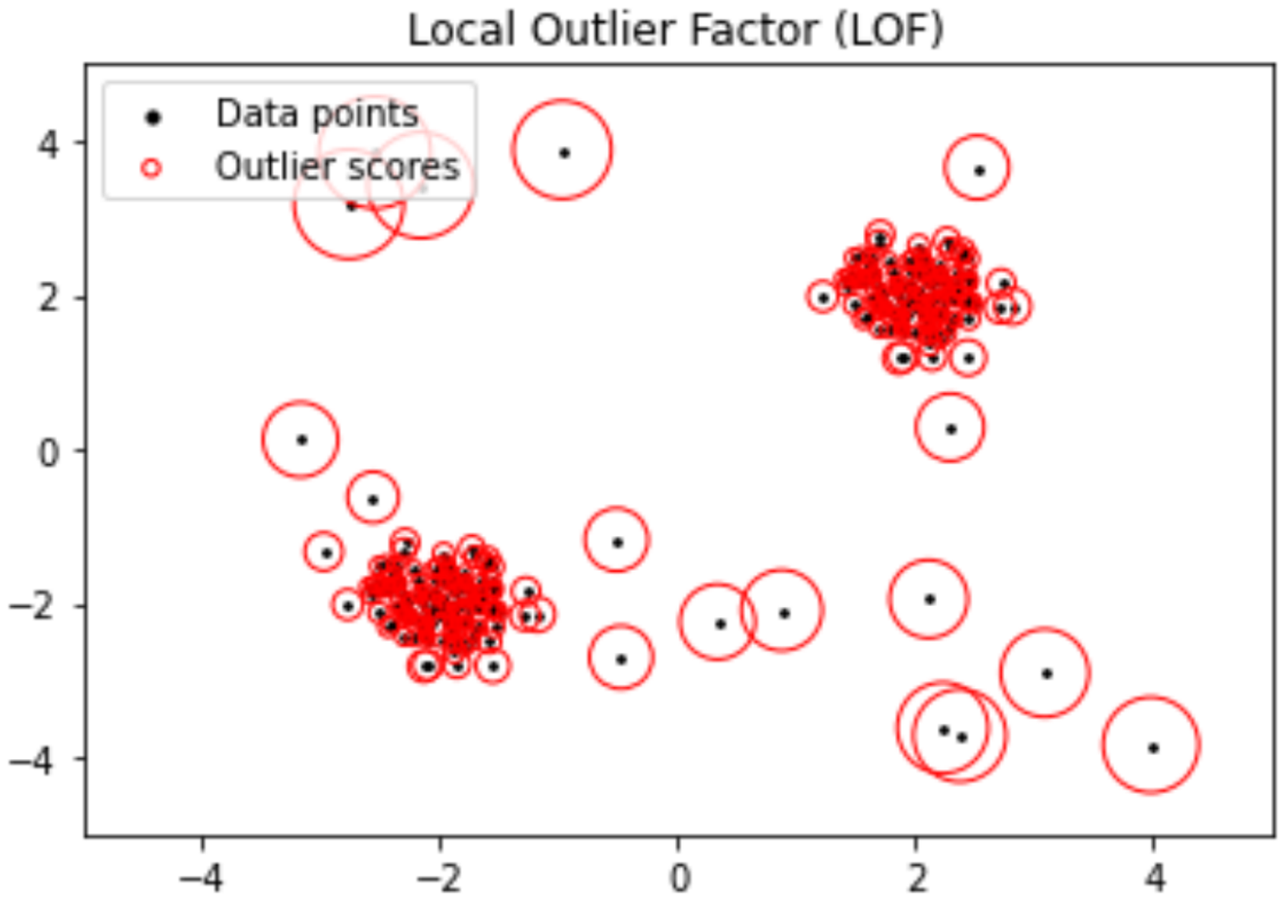
- Points are classified using a score called local outlier factor (LOF)
- LOF is based on the concept of local density
- Locality is chosen by setting a value to `n_neighbors` parameter
- Points with lower density are classified as outliers



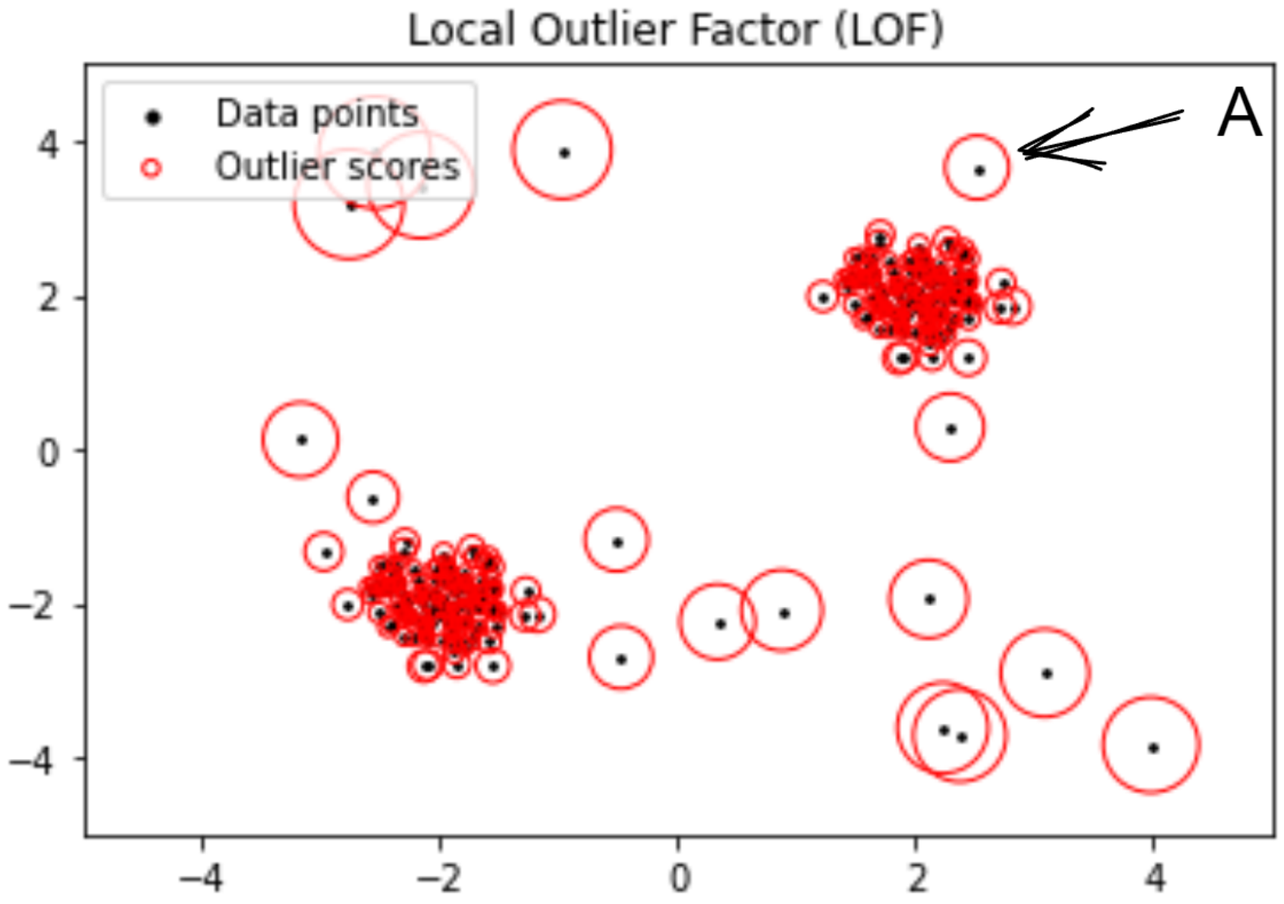
The importance of locality

- The word **local** is key
- LOF score is **only** compared to the score of nearby points

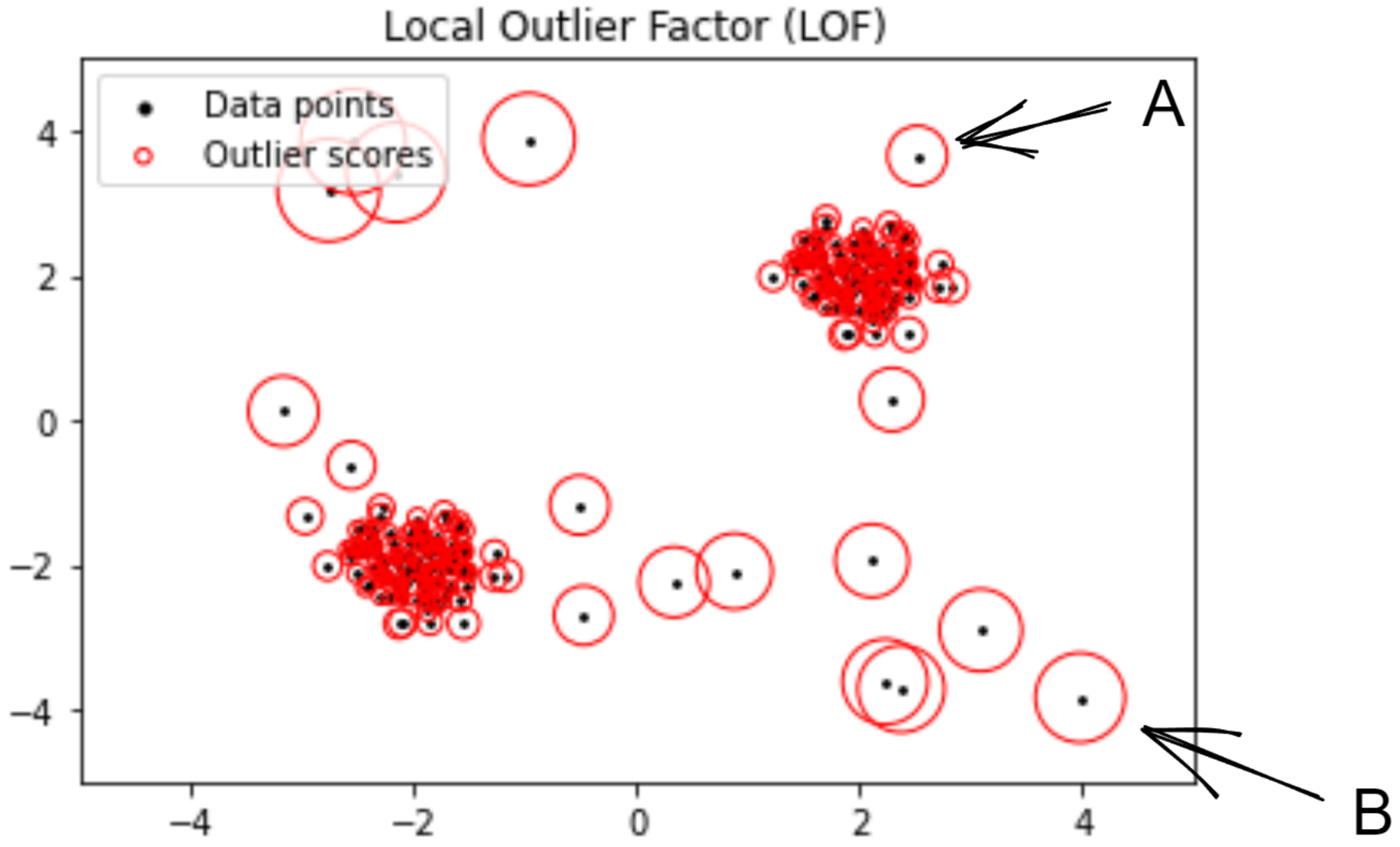
LOF visualized



LOF visualized



LOF visualized



Transformed dataset

```
import pandas as pd

males_transformed = pd.read_csv("males_transformed.csv")

males_transformed.head()
```

	abdominalextensiondepthsitting	acromialheight	acromionradialelength	...
0	0.365531	0.427976	0.113152	
1	-0.492137	-0.724973	-0.529301	
2	0.857097	-0.146048	0.357496	
3	-0.462610	-1.521525	-1.467860	
4	-0.026349	2.151957	1.985876	

LOF in action

```
from pyod.models.lof import LOF

# Fit
lof = LOF(n_neighbors=20, metric="manhattan")
lof.fit(males_transformed)

print(lof.labels_)
```

```
array([0, 0, 0, ..., 0, 0, 1])
```

Filtering in LOF

```
# Isolate the outliers
probs = lof.predict_proba(males_transformed)

is_outlier = probs[:, 1] > 0.55
outliers = males_transformed[is_outlier]

len(outliers)
```

```
2
```

LOF details

- `n_neighbors` is the most important
- Rule of thumb to tune `n_neigbors` :
  - 20 neighbors for <10% contamination
- No aggregation allowed:
  - `method` is always `largest`

LOF drawbacks

- Hard to interpret
- LOF score combines:
  - distance between points
  - reachability distance
  - many other components
- No specific range of LOF score for outliers

Let's practice!

ANOMALY DETECTION IN PYTHON