Mahalanobis Distance – Understanding the math with examples (python)

by Selva Prabhakaran

Mahalanobis distance is an effective multivariate distance metric that measures the distance between a point and a distribution. It is an extremely useful metric having, excellent applications in multivariate anomaly detection, classification on highly imbalanced datasets and one-class classification.

This post explains the intuition and the math with practical examples on three machine learning use cases.

Content

What doe… Global Interpreter Lock – …

P… – What does the yield … do?

…bda Function in Python – How …nd When to use?

Investor's Portfolio Optimization with Python

datetime in Python – Simplified Guide with Clear Examples

Python Collections – Complete Guide

pdb – How to use Python debugger

Python JSON – Guide

How to use tf.function to speed up Python code in Tensorflow

List Comprehensions in Python – My Simplified Guide

Mahalonobis Distance – Understanding the math with examples (python)

Parallel Processing in Python – A Practical Guide with Examples

Python @Property Explained – How to Use and When? (Full Examples)

Python Logging – Simplest Guide with Full Code and Examples

Python Regular Expressions Tutorial and Examples: A Simplified Guide

Requests in Python Tutorial – How to send HTTP requests in Python?

Simulated Annealing Algorithm Explained from Scratch

Setup Python environment for ML

Numpy.median() – How to compute median in Python

add Python to PATH – How to add Python to the PATH environment variable in Windows?

Install pip mac – How to install pip in MacOS?: A Comprehensive Guide

Install opencv python – A

1. Introduction

Mahalanobis distance is an effective multivariate distance metric that measures the distance between a point (vector) and a distribution. It has excellent applications in multivariate anomaly detection, classification on highly imbalanced datasets and one-class classification and more untapped use cases.

Considering its extremely useful applications, this metric is seldom discussed or used in stats or ML workflows. This post explains the why and the when to use Mahalanobis distance and then explains the intuition and the math with useful applications.

2. What's wrong with using Euclidean Distance for Multivariate data?

Let's start with the basics.

Euclidean distance is the commonly used straight line distance between two points. If the two points are in a two-dimensional plane (meaning, you have two numeric columns (p) and (q)) in your dataset), then the Euclidean distance between the two points (p1, q1) and (p2, q2) is:

$$d(p,q) = \sqrt{(p_1 - q_1)2 + (p_2 - q_2)2}$$

This formula may be extended to as many dimensions you want:

$$d(p,q) = \sqrt{(p_1 - q_1)2 + (p_2 - q_2)2 \ + \cdots + (p_n - q_n)2}$$

Gradient ... – A Concise Introd... ...n Scratch

P... ...ce Tradeoff – Clearly ...d

...et Package – A Practical Guide ...o Machine Learning in R

Linear Regression in Julia

ARIMA Model – Complete Guide to Time Series Forecasting in Python

How Naive Bayes Algorithm Works? (with example and full code)

Principal Component Analysis (PCA) – Better Explained

Mahalonobis Distance – Understanding the math with examples (python)

Investor's Portfolio Optimization with Python using Practical Examples

Augmented Dickey Fuller Test (ADF Test) – Must Read Guide

Complete Introduction to Linear Regression in R

Cosine Similarity – Understanding the math and how it works (with python codes)

Feature Selection – Ten Effective Techniques with Examples

Gensim Tutorial – A Complete Beginners Guide

K-Means Clustering Algorithm from Scratch

KPSS Test for Stationarity

Lemmatization Approaches with Examples in Python

Python Numpy – Introduction to ndarray [Part 1]

Numpy Tutorial Part 2 – Vital Functions for Data Analysis

P-Value – Understanding from Scratch

Vector Autoregression (VAR) –

I mean by that? Let's consider the following tables:

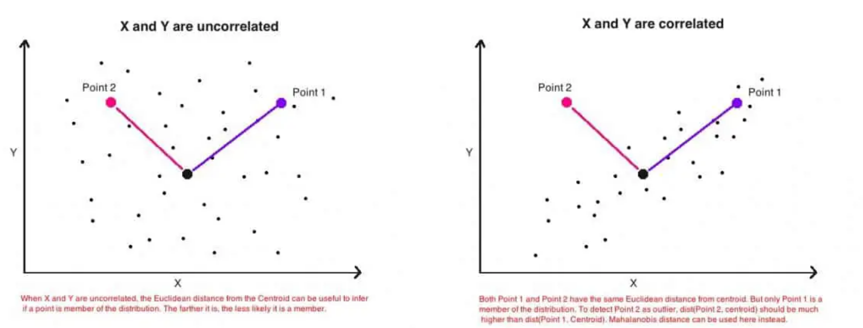
Area (sq.ft)	Price ($ 1000's)	Area (acre)	Price ($M)
2400	156000	0.0550944	156
1950	126750	0.0447642	126.75
2100	105000	0.0482076	105
1200	78000	0.0275472	78
2000	130000	0.045912	130
900	54000	0.0206604	54

The two tables above show the 'area' and 'price' of the same objects. Only the units of the variables change. Since both tables represent the same entities, the distance between any two rows, point A and point B should be the same. But Euclidean distance gives a different value even though the distances are technically the same in physical space.

This can technically be overcome by scaling the variables, by computing the z-score (ex: (x – mean) / std) or make it vary within a particular range like between 0 and 1.

But there is another major drawback.

That is, **if the dimensions (columns in your dataset) are correlated to one another, which is typically the case in real-world datasets, the Euclidean distance between a point and the center of the points (distribution) can give little or misleading information about how close a point really is to the cluster.**



The above image (on the right) is a simple scatterplot of two

Examples
Evalua... cs for Classification Mo... w to measure ...ce of machine learning ...s?
...pic modeling visualization – How to present the results of LDA models?
Topic Modeling with Gensim (Python)
Building chatbot with Rasa and spaCy
How to detect outliers using IQR and Boxplots?
How to detect outliers with z-score

equally distant (Euclidean) from the center. But only one of them (blue) is actually more close to the cluster, even though, technically the Euclidean distance between the two points are equal. This is because, Euclidean distance is a distance between two points only. It does not consider how the rest of the points in the dataset vary.

So, it cannot be used to really judge how close a point actually is to a distribution of points. What we need here is a more robust distance metric that is an accurate representation of how distant a point is from a *distribution*.

3. What is Mahalanobis Distance?

Mahalonobis distance is the distance between a point and a distribution. And not between two distinct points. It is effectively a multivariate equivalent of the Euclidean distance.

It was introduced by **Prof. P. C. Mahalanobis** in 1936 and has been used in various statistical applications ever since. However, it's not so well known or used in the machine learning practice. Well, let's get into it. So computationally, how is Mahalanobis distance different from Euclidean distance?

1. It transforms the columns into uncorrelated variables
2. Scale the columns to make their variance equal to 1
3. Finally, it calculates the Euclidean distance.

The above three steps are meant to address the problems with Euclidean distance we just talked about. But how? Let's look at the formula and try to understand its components.

4. The math and intuition behind Mahalanobis Distance

The formula to compute Mahalanobis distance is as follows:

$$D^2 = (x - m)^T \cdot C^{-1} \cdot (x - m)$$

```
where,
 - D^2        is the square of the Mahalanobis distanc
 - x          is the vector of the observation (row in
 - m          is the vector of mean values of independ
 - C^(-1)     is the inverse covariance matrix of inde
```

So, how to understand the above formula? Let's take the (x – m)^T . C^(-1) term. (x – m) is essentially the distance of the vector from the mean. We then divide this by the covariance matrix (or multiply by the inverse of the covariance matrix). If you think about it, this is essentially a multivariate equivalent of the regular standardization (z = (x – mu)/sigma).

That is, z = (x vector) – (mean vector) / (covariance matrix).

So, What is the effect of dividing by the covariance? If the variables in your dataset are strongly correlated, then, the covariance will be high. Dividing by a large covariance will effectively reduce the distance.

Likewise, if the X's are not correlated, then the covariance is not high and the distance is not reduced much. So effectively, it addresses both the problems of scale as well as the correlation of the variables that we talked about in the introduction.

5. How to compute Mahalanobis Distance in Python

```
filepath = 'https://raw.githubusercontent.com/selva86/datasets,
df = pd.read_csv(filepath).iloc[:, [0,4,6]]
df.head()
```

	carat	depth	price
0	0.23	61.5	326
1	0.21	59.8	326
2	0.23	56.9	327
3	0.29	62.4	334
4	0.31	63.3	335

Let's write the function to calculate Mahalanobis Distance.

```
def mahalanobis(x=None, data=None, cov=None):
    """Compute the Mahalanobis Distance between each row of x a
    x    : vector or matrix of data with, say, p columns.
    data : ndarray of the distribution from which Mahalanobis d
    cov  : covariance matrix (p x p) of the distribution. If No
    """
    x_minus_mu = x - np.mean(data)
    if not cov:
        cov = np.cov(data.values.T)
    inv_covmat = sp.linalg.inv(cov)
    left_term = np.dot(x_minus_mu, inv_covmat)
    mahal = np.dot(left_term, x_minus_mu.T)
    return mahal.diagonal()

df_x = df[['carat', 'depth', 'price']].head(500)
df_x['mahala'] = mahalanobis(x=df_x, data=df[['carat', 'depth'
df_x.head()
```

	carat	depth	price	mahala
0	0.23	61.5	326	1.709860
1	0.21	59.8	326	3.540097
2	0.23	56.9	327	12.715021
3	0.29	62.4	334	1.454469
4	0.31	63.3	335	2.347239

6. Usecase 1: Multivariate outlier detection using Mahalanobis distance

Assuming that the test statistic follows chi-square distributed with 'n' degree of freedom, the critical value at a 0.01 significance level and 2 degrees of freedom is computed as:

```
# Critical values for two degrees of freedom
from scipy.stats import chi2
chi2.ppf((1-0.01), df=2)
#> 9.21
```

That mean an observation can be considered as extreme if its Mahalanobis distance exceeds 9.21. If you prefer P values instead to determine if an observation is extreme or not, the P values can be computed as follows:

```
# Compute the P-Values
df_x['p_value'] = 1 - chi2.cdf(df_x['mahala'], 2)

# Extreme values with a significance level of 0.01
```

X

	carat	depth	price	mahala	p_value
2	0.23	56.9	327	12.715021	0.001734
91	0.86	55.1	2757	23.909643	0.000006
97	0.96	66.3	2759	11.781773	0.002765
172	1.17	60.2	2774	9.279459	0.009660
204	0.98	67.9	2777	20.086616	0.000043
221	0.70	57.2	2782	10.405659	0.005501
227	0.84	55.1	2782	23.548379	0.000008
255	1.05	65.8	2789	11.237146	0.003630
284	1.00	58.2	2795	10.349019	0.005659
298	1.01	67.4	2797	17.716144	0.000142

If you compare the above observations against rest of the dataset, they are clearly extreme.

7. Usecase 2: Mahalanobis Distance for Classification Problems

Mahalanobis distance can be used for classification problems. A naive implementation of a Mahalanobis classifier is coded below. The intuition is that, an observation is assigned the class that it is closest to based on the Mahalanobis distance. Let's see an example implementation on the `BreastCancer` dataset, where the objective is to determine if a tumour is benign or malignant.

```
 df = pd.read_csv('https://raw.githubusercontent.com/selva86/d
                  usecols=['Cl.thickness', 'Cell.size', 'Marg.ad
                           'Epith.c.size', 'Bare.nuclei', 'Bl.c
                           'Mitoses', 'Class'])

df.dropna(inplace=True)  # drop missing values.
df.head()
```

	Cl.thickness	Cell.size	Marg.adhesion	Epith.c.size	Bare.nuclei	Bl.cromatin	Normal.nucleoli	Mitoses	Class
0	5	1	1	2	1.0	3	1	1	0
1	5	4	5	7	10.0	3	2	1	0
2	3	1	1	2	2.0	3	1	1	0
3	6	8	1	3	4.0	3	7	1	0
4	4	1	3	2	1.0	3	1	1	0

Breast Cancer Dataset

measure the Mahalanobis distances between a given observation (row) and both the positive (`xtrain_pos`) and negative datasets(`xtrain_neg`). Then that observation is assigned the class based on the group it is closest to.

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(df.drop('Class

# Split the training data as pos and neg
xtrain_pos = xtrain.loc[ytrain == 1, :]
xtrain_neg = xtrain.loc[ytrain == 0, :]
```

Let's build the `MahalanobiBinaryClassifier` . To do that, you need to define the `predict_proba()` and the `predict()` methods. This classifier does not require a separate `fit()` (training) method.

```
class MahalanobisBinaryClassifier():
    def __init__(self, xtrain, ytrain):
        self.xtrain_pos = xtrain.loc[ytrain == 1, :]
        self.xtrain_neg = xtrain.loc[ytrain == 0, :]

    def predict_proba(self, xtest):
        pos_neg_dists = [(p,n) for p, n in zip(mahalanobis(xte
        return np.array([(1-n/(p+n), 1-p/(p+n)) for p,n in pos_

    def predict(self, xtest):
        return np.array([np.argmax(row) for row in self.predic


clf = MahalanobisBinaryClassifier(xtrain, ytrain)
pred_probs = clf.predict_proba(xtest)
pred_class = clf.predict(xtest)

# Pred and Truth
pred_actuals = pd.DataFrame([(pred, act) for pred, act in zip(
print(pred_actuals[:5])
```

X

```
    pred  true
0      0     0
1      1     1
2      0     0
3      0     0
4      0     0
```

Let's see how the classifier performed on the test dataset.

```
 from sklearn.metrics import classification_report, accuracy_s
truth = pred_actuals.loc[:, 'true']
pred = pred_actuals.loc[:, 'pred']
scores = np.array(pred_probs)[:, 1]
print('AUROC: ', roc_auc_score(truth, scores))
print('\nConfusion Matrix: \n', confusion_matrix(truth, pred))
print('\nAccuracy Score: ', accuracy_score(truth, pred))
print('\nClassification Report: \n', classification_report(tru
```

Output:

```
 AUROC:  0.9909743589743589

Confusion Matrix:
 [[113  17]
 [  0  75]]

Accuracy Score:  0.9170731707317074

Classification Report:
              precision    recall  f1-score   support


          0       1.00      0.87      0.93       130
          1       0.82      1.00      0.90        75


avg / total       0.93      0.92      0.92       205
```

Mahalanobis distance alone is able to contribute to this much

X

One Class classification is a type of algorithm where the training dataset contains observations belonging to only one class.

With only that information known, the objective is to figure out if a given observation in a new (or test) dataset belongs to that class. You might wonder when would such a situation occur.

Well, it's a quite common problem in Data Science.

For example consider the following situation: You have a large dataset containing millions of records that are NOT yet categorized as 1's and 0's. But you also have with you a small sample dataset containing only positive (1's) records. By learning the information in this sample dataset, you want to classify all the records in the large dataset as 1's and 0's. Based on the information from the sample dataset, it is possible to tell if any given sample is a 1 or 0 by viewing only the 1's (and having no knowledge of the 0's at all). This can be done using Mahalanobis Distance.

Let's try this on the `BreastCancer` dataset, only this time we will consider only the malignant observations (class column=1) in the training data.

```
  df = pd.read_csv('https://raw.githubusercontent.com/selva86/d
                   usecols=['Cl.thickness', 'Cell.size', 'Marg.a
                            'Epith.c.size', 'Bare.nuclei', 'Bl.c
                            'Mitoses', 'Class'])

 df.dropna(inplace=True)  # drop missing values.
```

X

Splitting 50% of the dataset into training and test. Only the 1's are retained in the training data.

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(df.drop('Class

# Split the training data as pos and neg
xtrain_pos = xtrain.loc[ytrain == 1, :]
```

Let's build the `MahalanobisOneClassClassifier` and get the mahalanobis distance of each datapoint in x from the training set (`xtrain_pos`).

```
class MahalanobisOneclassClassifier():
    def __init__(self, xtrain, significance_level=0.01):
        self.xtrain = xtrain
        self.critical_value = chi2.ppf((1-significance_level),
        print('Critical value is: ', self.critical_value)

    def predict_proba(self, xtest):
        mahalanobis_dist = mahalanobis(xtest, self.xtrain)
        self.pvalues = 1 - chi2.cdf(mahalanobis_dist, 2)
        return mahalanobis_dist

    def predict(self, xtest):
        return np.array([int(i) for i in self.predict_proba(xt

clf = MahalanobisOneclassClassifier(xtrain_pos, significance_l
```

```
mdist_actuals = pd.DataFrame([(m, act) for m, act in zip(mahal
print(mdist_actuals[:5])
```

```
Critical value is:  14.067140449340169
      mahal  true_class
0  13.104716           0
1  14.408570           1
2  14.932236           0
3  14.588622           0
4  15.471064           0
```

We have the Mahalanobis distance and the actual class of each observation. I would expect those observations with low Mahalanobis distance to be 1's. So, I sort the `mdist_actuals` by Mahalanobis distance and quantile cut the rows into 10 equal sized groups. The observations in the top quantiles should have more 1's compared to the ones in the bottom. Let's see.

```
# quantile cut in 10 pieces
mdist_actuals['quantile'] = pd.qcut(mdist_actuals['mahal'],
                                    q=[0, .10, .20, .3, .4, .5
                                    labels=[1, 2, 3, 4, 5, 6, 

# sort by mahalanobis distance
mdist_actuals.sort_values('mahal', inplace=True)
perc_truths = mdist_actuals.groupby('quantile').agg({'mahal': 
print(perc_truths)
```

```
          avg_mahaldist  sum_of_trueclass
quantile
1              3.765496                33
2              6.511026                32
3              9.272944                30
4             12.209504                20
5             14.455050                 4
```

```
9            21.533159            2
10           23.524055            1
```

If you notice above, nearly 90% of the 1's (malignant cases) fall within the first 40%ile of the Mahalanobis distance.

Incidentally, all of these are lower than the critical value pf 14.05. So, let's the critical value as the cutoff and mark those observations with Mahalanobis distance less than the cutoff as positive.

```
from sklearn.metrics import classification_report, accuracy_s

# Positive if mahalanobis
pred_actuals = pd.DataFrame([(int(p), y) for y, p in zip(ytest

# Accuracy Metrics
truth = pred_actuals.loc[:, 'true']
pred = pred_actuals.loc[:, 'pred']
print('\nConfusion Matrix: \n', confusion_matrix(truth, pred))
print('\nAccuracy Score: ', accuracy_score(truth, pred))
print('\nClassification Report: \n', classification_report(tru
```

```
 Confusion Matrix:
 [[183  29]
 [ 15 115]]

Accuracy Score:  0.8713450292397661

Classification Report:
              precision    recall  f1-score   support

          0       0.92      0.86      0.89       212
          1       0.80      0.88      0.84       130

avg / total       0.88      0.87      0.87       342
```