Unsupervised Anomaly Detection – Kaggle 실습

<https://www.kaggle.com/victorambonati/unsupervised-anomaly-detection/notebook#2-Models>

**[Motivation]**

Dataset : NAB datasets

시각화 – 알고리즘이 어떤 것을 detect하는지를 이해하기 위해서.

**[알고리즘]**

* Cluster based anomaly detection (K-mean)
* Repartition of data into categories then Gaussian/Elliptic Envelope on each categories separately
* Markov Chain
* Isolation Forest
* RNN (comparison between prediction and reality)

(Unsupervised Anomaly Detection – Algorithm.docx 참고!)

**[라이브러리]**

pandas, numpy

matplotlib, seaborn (시각화)

sklearn (기계학습 알고리즘)

**[Functions]**

|  |  |
| --- | --- |
| getDistanceByPoint(data, model) | 각 점과 가장 가까운 centroid 사이의 거리의 Series를 return |
| getTransitionMatrix(df) | Transition matrix를 얻기 위해 markov model을 train |
|  |  |
|  |  |
| anom |  |
| markovAnomaly(df, windows\_size, threshold) |  |

**1 Data**

* 1. **Extract data**

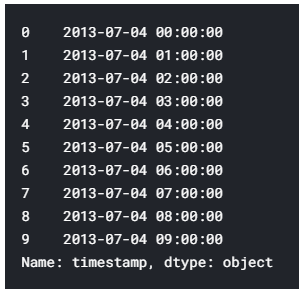
NAB – realKnownCause/ambient\_temperature\_system\_failure.csv

df = pd.read\_csv

* 1. **Understand data**

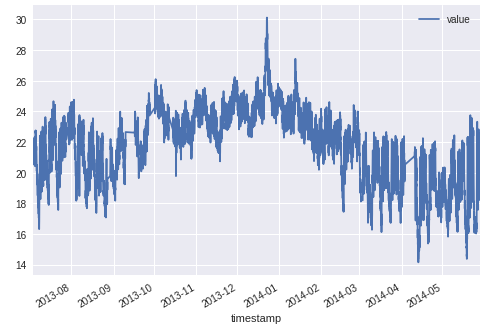
7267개 데이터, 2columns (timestamp, value)

timestamp



Value의 평균값 : 71.2424…

Value의 fahrenheit을 ‘C로 바꾼 뒤 시각화

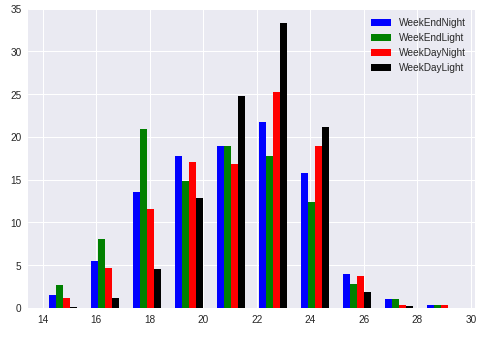


**1.3 Feature engineering**

유용한 feature들을 추출(extract)해낸다.

‘hours’, ‘daylight’, ‘DayOfTheWeek’, ‘WeekDay’

4개의 카테고리 생성 – week end/day week & night/day



평일의 낮(daylight of business day)에 온도가 더 stable하다.

(왜냐하면 같은 온도를 가진 데이터가 많으니까, 평균에 모여 있으니까)

1. **Models**
   1. **Cluster only**

**-Use for collective anomalies (unordered)**

feature들의 **usual combination**을 grouping했다. (We group together the usual combination of features.)

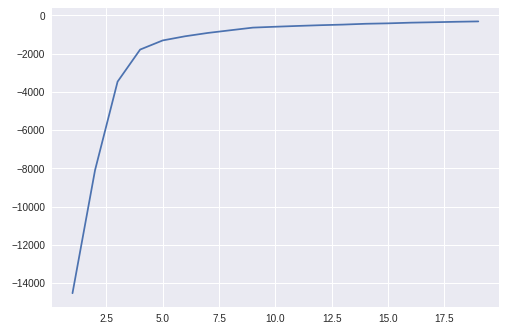
클러스터에서 먼 point일수록 feature의 usual combination과 함께 있는 point이고, 그러한 point들을 anomaly라고 여긴다.

(The points that are far from the cluster are points with usual combination of features. We consider those points as anomalies)

|  |  |  |  |
| --- | --- | --- | --- |
| # useful feature들을 take하고 그들을 표준화(standardize)한다.   |  | | --- | | data = df[[‘value’, ‘hours’, ‘daylight’, ‘DayOfTheWeek’, ‘WeekDay’]]  min\_max\_scaler = preprocessing.StandardScaler()  np\_scaled = min\_max\_scaler.fit\_transform(data)  data = pd.DataFrame(np\_scaled) |  * sklearn.preprocessing.StandardScaler() * fit\_transform()   # 2 중요 feature들을 reduce한다.   |  | | --- | | pca = PCA(n\_components=2)  data = pca.fit\_transform(data) |  * PCA   # 2 새로운 feature들을 standardize한다.   |  | | --- | | min\_max\_scaler = preprocessing.StandardScaler()  np\_scaled = min\_max\_scaler.fit\_transform(data)  data = pd.DataFrame(np\_scaled) | |

# 다른 수의 centroid들로 계산한다. Loss plot(elbow method)를 보기 위해서

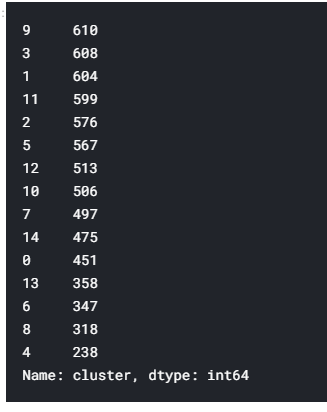
(k-means에서 centroid의 개수(k의 개수) 정하기, loss 최소인거)



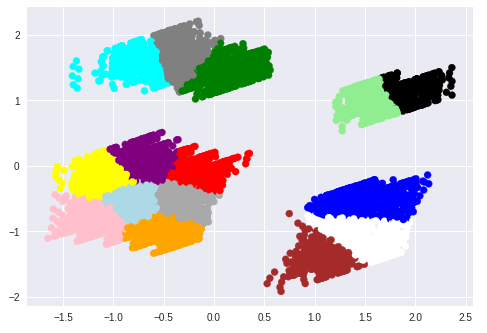
# **15**개의 centroid들을 임의로 고름. 그리고 central dataframe에 데이터를 추가

df[‘principal\_feature1’] = data[0]

df[‘principal\_feature2’] = data[1]



# 2개의 main feature들로 다른 cluster를 plot함. (15개, 0~14)

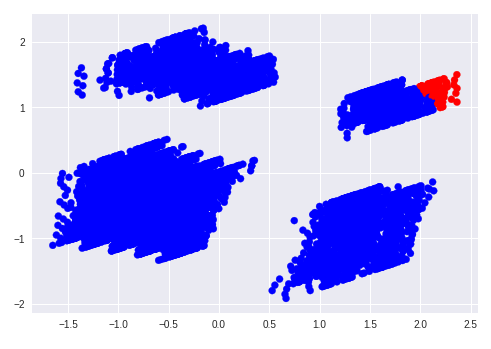


#각 point에서 가장 가까운 centroid와의 거리를 구한다. 가장 큰 거리가 anomaly로 여겨진다.

df[‘anomaly21’] = (distance >= threshold).astype(int)

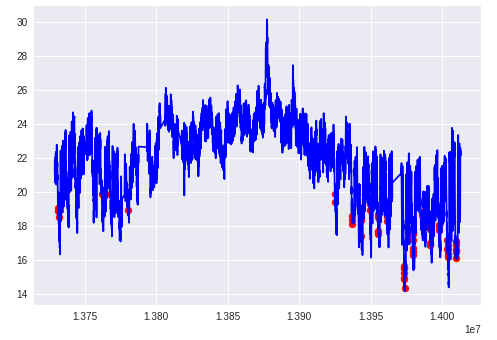
# cluster view를 통해 anomaly를 visualization

(red가 anomaly)

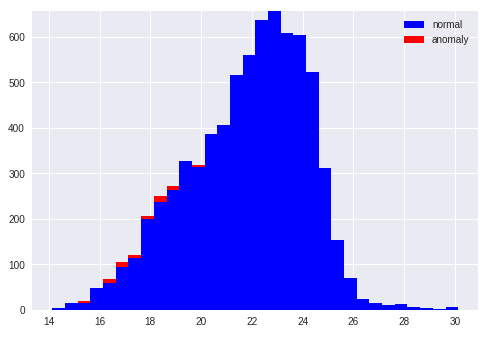


# anomaly **throughout time**의 visualization

df.loc



# anomaly with temperature repartition의 visualization



클러스터 방법은 unusually low한 기록의 끝부분 주위의 낮은 온도를 detect한다.

(Cluster method detects the low temperature around the end of record as unusually low.)

가장 높은 온도를 detect하지는 못한다.

* 1. **Categories + Gaussian**

**문맥적인 데이터(contextual data)와 collective한 어노멀리(unordered)를 위해 사용**

데이터를 (우리가 생각하는) 중요 카테고리로 나눈다. 혹은 다른 클러스터를 기반으로 데이터를 구분할 수도 있다. 그 후에 카테고리를 통해 독립적으로 아웃라이어(gaussian repartition, unimodal)을 찾을 수 있다.

* gaussian repartition
* unimodal : 최댓값이 오직 하나만 있는 것. (그래프의 모양이 단봉형)

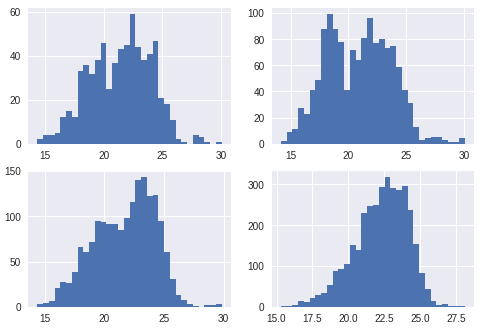
#이전에 정의된 카테고리를 바탕으로 4가지의 서로 다른 data set을 만든다.

|  |
| --- |
| df\_class0 = df.loc[df[‘categories’] == 0, ‘value’] |

* df.loc

# 카테고리로 temperature repartition을 plot한다.

|  |
| --- |
| fig, axs = plt.subplots(2, 2)  df\_class0.hist(ax=axs[0,0],bins=32)  … |



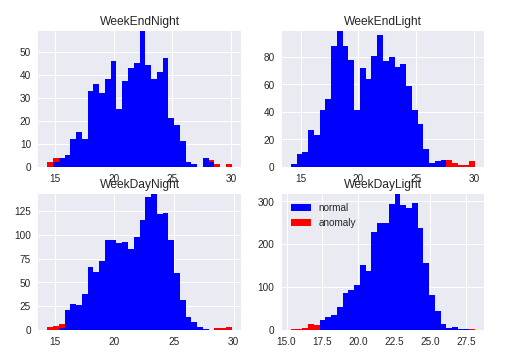
#각 카테고리에 ellipticEnvelope(gaussian distribution)을 적용한다.

|  |
| --- |
| envelope = EllipticEnvelope(contamination = outliers\_fraction)  X\_train = df\_class0.values.reshape(-1, 1)  envelope.fit(X\_train)  df\_class0 = pd.DataFrame(df\_class0)  df\_class0[‘deviation’] = envelope.decision\_function(X\_train)  df\_class0[‘anomaly’] = envelope.predict(X\_train)  … |

* EllipticEnvelope

# anomalies가 있는 카테고리로 temperature repartition을 plot한다.

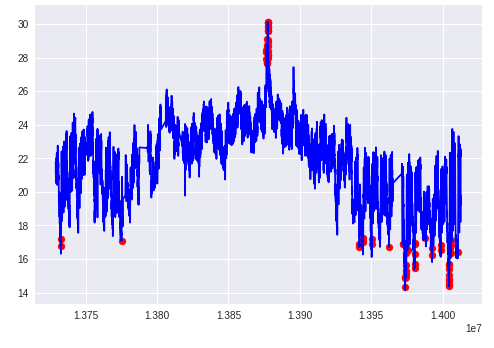
|  |
| --- |
| a0 = df\_class0.loc[df\_class0[‘anomaly’] == 1, ‘value’]  b0 = df\_class0.loc[df\_class0[‘anomaly’] == -1, ‘value’] |



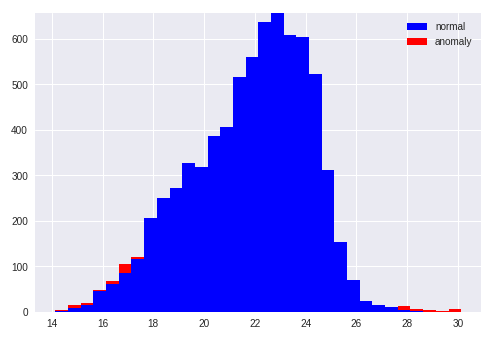
# 메인에 데이터를 추가한다.

# 어노멀리를 시간으로 visualization (throughout time) (viz 1)

‘time\_epoch’



# 어노멀리 시각화 with temperature repartition (viz 2)



* extreme values에 좋은 detection. 문맥 separation이 detection에 정확성을 더해준다.
  1. **Cluster + Gaussian**

2.2의 solution과 비슷하지만 cluster로 나눈다.

* 1. **Markov chains**

**sequential 어노멀리(ordered)**를 위해 사용

마코프 체인의 상태를 정의하는 데이터 포인트의 분리가 필요하다. (We need discretize the data points in define states for markov chain.)

이 예제에서 state를 정의하기 위해 ‘value’만 채택한다. 그리고 5개 레벨의 value를 정의한다. (very low / low / average / high / very high)/(VL, L, A, H, VH)

마코프 체인은 sequence의 probability를 (VL, L, L, A, A, L, A)이런 식으로 계산할 것이다. 만약 확률이 매우 약하다면(weak) 그 sequence를 anomaly라고 여긴다.

|  |
| --- |
| # 각각의 state에 대해서 정의한다.  x1 = (df[‘value’] <=18).astype(int)  x2 = ((df[‘value’] > 18) & (df[‘value’]<=21)).astype(int)  …  df\_mm = x1 + 2\*x2 + 3\*x3 + 4\*x4 + 5\*x5  # 데이터셋을 위해 anomaly labels를 get한다. (5개 value의 sequence와 anomaly를 평가하기 위해서 = 20%미만의 probable)  # NOT AVAILABLE IN KAGGLE KERNEL- pyemma를 사용함.  # df\_anomaly = markovAnomaly(df\_mm, 5, 0,20)  # df\_anomaly = pd.Series(df\_anomaly)  # print(df\_anomaly.value\_counts()) |

|  |
| --- |
| “””  # main에 data를 추가한다.  df[‘anomaly24’] = df\_anomaly  #anomaly 비주얼화(visualization) – throughout time(viz 1)  fig, ax = plt.subplots()  a = df.loc[df[‘anomaly24’] == 1, (‘time\_epoch’, ‘value’)] # anomaly  ax.plot(df[‘time\_epoch’], df[‘value’], color=’blue’)  ax.scatter(a[‘time\_epoch’], a[‘value’], color=’red’)  plt.show()  “”” |

out

|  |
| --- |
| “””  # 어노멀리 비주얼화 with temperature repartition(viz 2)  a = df.loc[df[‘anomaly24’] == 0, ‘value’]  b = df.loc[df[‘anomaly24’] == 1, ‘value’]  fig, axs = plt.subplots()  axs.hist([a, b], bins=32, stacked=True, color[‘blue’, ‘red’])  plt.legent()  plt.show()  “”” |

out

* unusual한 sequence를 찾을 수 있지만 extreme value는 찾지 못한다. 이 예제에서 연관성(relevance)을 평가하기에 더 힘들었다. Some interesting cycle에는 sequence size (5)가 매치가 될 것이다.
  1. Isolation Forest

collective anomaly(unordered)를 위해 사용한다.

간단하고, different data repartition에 잘 동작한다. 그리고 high dimention data에 효과적이다.

|  |
| --- |
| #유용한 feature을 모으고, 정규화한다.(standardize)  data = df[[‘value’, ‘hours’, ‘daylight’, ‘DayOfTheWeek’, ‘WeekDay’]]  min\_max\_scaler = preprocessing.StandardScaler()  np\_scaled = min\_max\_scaler.fit\_transform(data)  data = pd.DataFrame(np\_scaled)  #Isolation forest train을 한다.  mode = IsolationForest(contamination = outliers\_fraction) ## 0.1  model.fit(data)  #main에 데이터를 추가한다.  df[‘anomaly25’] = pd.Series(model.predict(data))  df[‘anomaly25’] = df[‘anomaly25’].map( {1: 0, -1: 1} )  print(df[‘anomaly25’].value\_counts()) |

