Koneoppiminen

Nimi: Petri Salminen
Opnum: 243840
Email: petri.salminen@tuni.fi

Ladataan ensiksi viime tehtävässä talletetut datat ja tarvittavat kirjastot

In [1]:
```
import pickle
import pandas as pd
import holoviews as hv
hv.extension('matplotlib')
%matplotlib inline


with open('df.pickle', 'rb') as f:
    df = pickle.load(f)
```

In [2]:
```
df.head()
```

Out[2]:

	IP	Time	URL	Status	RequestType	HttpVersion	URL2
0	176161281	2017-11-29 06:58:55	/login.php	200	GET	1.1	/login.php
1	176161281	2017-11-29 06:59:02	/process.php	302	POST	1.1	/process.php
2	176161281	2017-11-29 06:59:03	/home.php	200	GET	1.1	/home.php
3	176357889	2017-11-29 06:59:04	/js/vendor /moment.min.js	200	GET	1.1	/js/vendor /moment.min.js
4	176292353	2017-11-29 06:59:06	/bootstrap-3.3.7/js /bootstrap.js	200	GET	1.1	/bootstrap-3.3.7/js /bootstrap.js

Jeps vaikuttaa siltä että pikkelöinti ja sieltä datan noutaminen tapahtui yllättävän näppärästi! Henkilökohtaisesti suosittelen kyseistä kirjastoa etenkin silloin, jos hyödynnetään ulkopuolisia rajapintoja. Devaillessa voi pikkelöidä parit pyynnöt ja sen jälkeen devailla niillä, kunnes on aika hakea lisää.

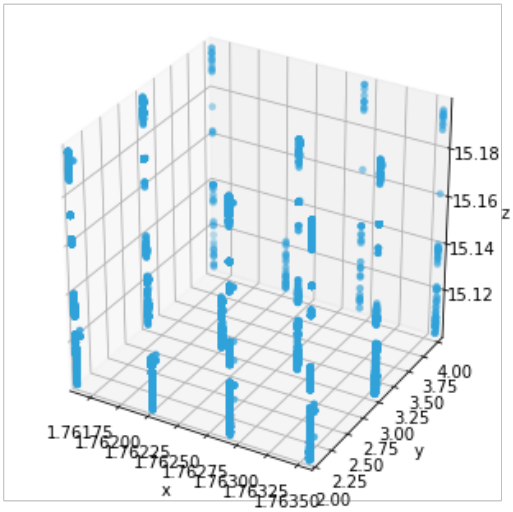
Seuraavaksi kokeilemme KPrototypes-algoritmia, joka on sekoitus KMeans ja KModes -algoritmeja. Kyseisen kirjaston lähdekoodi on saatavilla githubissa (https://github.com/nicodv/kmodes).

Hyödynnämme myös deepcopya, joka varmistaa, ettei alkuperäiseen df-muuttujaan kosketa. `HUOM! Deepcopy on harvinaisen raskas operaatio, joten sitä ei kannata käyttää missään loopeissa yms.`

In [51]:
```
from kmodes.kprototypes import KPrototypes
from copy import deepcopy
df_altered = deepcopy(df).dropna()
df_altered["Timestamp"] = (df["Time"] - pd.Timestamp("1970-01-01")) // pd.Timed
elta('1s') # Use unix timestamps

#df_altered[["IP", "Status", "Timestamp"]] = scaler.fit_transform(df_altered[["
IP", "Status", "Timestamp"]])
df_altered["Status"] = df_altered["Status"].apply(lambda x: x*10**6) # Make the
status as big as Timestamp and IP

n=3 # 1 for speed
print("=== {} ===".format(n))
kproto = KPrototypes(n_clusters=n, n_jobs=2, n_init=5, max_iter=1000, init='Cao
', verbose=2)
clusters = kproto.fit_predict(df_altered[["IP","Status", "Timestamp","URL2","Re
questType"]], categorical=[3,4])
print(kproto.cluster_centroids_)
print("")
# Print training statistics
print(kproto.cost_)
print(kproto.n_iter_)
scatter = hv.Scatter3D((df_altered["IP"] / 10**8, df_altered["Status"] / 10**8,
df_altered["Timestamp"] / 10**8))
scatter
```

```
=== 3 ===
Best run was number 1
[array([[1.76275022e+08, 2.00027840e+08, 1.51399186e+09],
       [1.76270914e+08, 4.04000000e+08, 1.51284843e+09],
       [1.76271839e+08, 3.02318336e+08, 1.51475036e+09]]), array([['/login.php'
, 'GET'],
       ['/login.php', 'GET'],
       ['/login.php', 'GET']], dtype='<U10')]

1.4198779337609659e+17
1
```

Out[51]:



Datasta löydettiin kolme klusteria, joita en kylläkään ymmärrä kyseisen 3d-scatterplotinkaan avulla. Lisäksi kategoriset muuttujat ovat kaikissa klustereissa samat, mikä saa minut kyseenalaistamaan tämänhetkiset algoritmin asetukset.

Kokeillaampa siis jotain täysin muuta, koska eihän tuosta ollut mitään apua kenellekkään. Kokeillaan rakentaa malli, joka ennustaa HTTP-pyynnön Status-muuttujan arvon. Aloitetaan jakamalla datasetti harjoitus- ja testaussetteihin. Sen jälkeen harjoitetaan malli ja kokeillaan, kuinka tarkka se oli.

In [5]:
```
from sklearn.model_selection import train_test_split
X = deepcopy(df)

X["Timestamp"] = (df["Time"] - pd.Timestamp("1970-01-01")) // pd.Timedelta('1s'
) # Use unix timestamps
y = df["Status"]

X_train, X_test, y_train, y_test = train_test_split(X[["IP", "Timestamp"]],y, t
est_size=0.25)
```

In [21]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred = gnb.predict(X_test)

from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[21]:
```
0.7183383991894631
```

Tarkkuus on 71.8%, mikä itseni yllätti positiivisesti. Katsotaanpas kuitenkin varmuuden vuoksi, kuinka paljon testisetissä on Status=200 -arvoja.

In [22]:
```
print(y_test.value_counts()[200] / y_test.size)
```

```
0.7183383991894631
```

Vai niin.. Kehitin juuri mallin, joka osaa sanoa, että vastaus (kaikkeen) on 200. Life well spent! Kokeillaanpas jotain, mistä saattaisi olla edes jotain hyötyä. LocalOutlierFactor:llä pyritään siihen, että saadaan matemaattinen malli nykyiselle "hyvälle" datasetille, jolloin voidaan sen avulla tunnistaa sellaiset uudet alkiot, jotka eivät ole "hyviä". Esimerkkitapauksessamme opetetaan mallille, että nykyiset lokitiedot ovat hyviä ja jatkossa pitää tunnistaa, vastaako alkio nykyisiä lokitietoja vai jotain muuta.

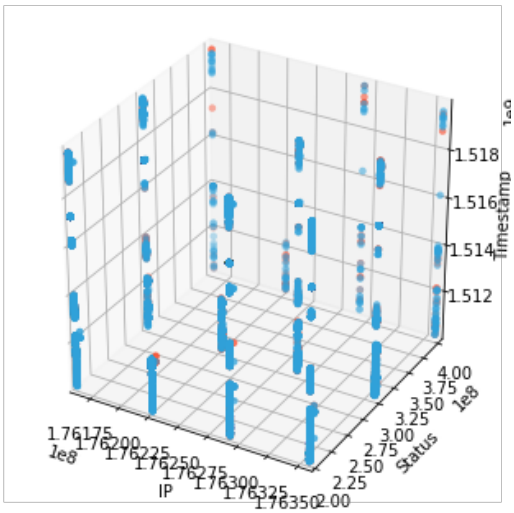
In [78]:
```
from sklearn.neighbors import LocalOutlierFactor
X_train, X_test = train_test_split(df_altered[["IP","Status", "Timestamp"]], te
st_size=0.25)
detector = LocalOutlierFactor(novelty=True, contamination="auto")
detector.fit(X_train)
pred = detector.predict(X_test)
```

```
/home/petri/jupyter/jupyter_env/lib/python3.6/site-packages/sklearn/neighbors/l
of.py:236: FutureWarning: default contamination parameter 0.1 will change in ve
rsion 0.22 to "auto". This will change the predict method behavior.
  FutureWarning)
```

Kokeillaan 3D-scatterplotilla, onko koulutus- ja testidatalla jotain silminnähtävää eroa.

In [79]:
```
scatter_train = hv.Scatter3D(X_train[["IP", "Status", "Timestamp"]])
scatter_test = hv.Scatter3D(X_test[["IP", "Status", "Timestamp"]])
scatter_train * scatter_test
```

Out[79]:

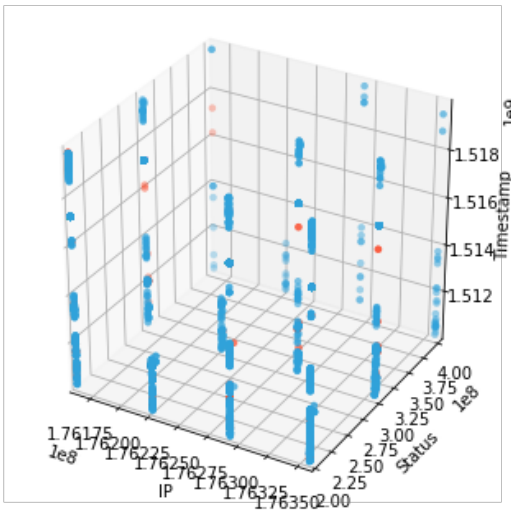


Eroa ei pahemmin näytä olevan. `HUOM! Hieman näkyy punaisia testitäpliä, mutta erittäin vähän` .

Kokeillaan nyt, miltä näyttää kun erotellaan vanhojen lokitietojen kaltaisiksi ja erilaisiksi luokitellut eri värillä.

In [87]:
```
scatter_outliers = hv.Scatter3D(X_test[pred==-1][["IP", "Status", "Timestamp"]]
)
scatter_buddies = hv.Scatter3D(X_test[pred==1][["IP", "Status", "Timestamp"]])
scatter_buddies * scatter_outliers
```

Out[87]:



Muutamat läheisistä palluroista irralliset näkyvät erittäin selkeästi.

Seuraavaksi testataan vielä, onko saatavilla olevilla tehoilla mahdollista ottaa myös URL mukaan OneHotEncoderin läpi. Tämä tarkoittaa sitä, että jokaisesta erilaisesta URL2-muuttujasta luodaan oma sarake dataframeen.

In [184]:
```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(sparse=False) # Sparse tarkoittaa, että sisällytetään myös
nollat. En tiedä onko tämä tarpeellista
enc.fit(df_altered["URL2"].values.reshape(-1, 1))
df_url_OH_encoded = pd.DataFrame(enc.transform(df_altered["URL2"].values.reshap
e(-1,1)), columns=enc.get_feature_names())
```

In [211]:
```
large_df = pd.concat([df_altered, df_url_OH_encoded], axis=1) # Yhdistetään df_
altered ja uusi df_url_OH_encoded
```

In [212]:
```
almost_as_large_df = large_df.drop(['Time','URL', 'RequestType', 'URL2'],axis=1
) # Pudotetaan turhat kategoriset pois
```

Seuraavaksi skaalataan muuttujat niin, että ne on sovitettu normaalijakaumaan. Tämä eliminoi sen, LocalOutlierFactor minimoi virhettä, joka on luonnollisesti suurempi luonnostaan suuremmilla luvuilla.

In [256]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

almost_as_large_array = scaler.fit_transform(almost_as_large_df)
std_df = pd.DataFrame(almost_as_large_array, columns=almost_as_large_df.columns
).dropna()
std_df.columns
```

Out[256]:
```
Index(['IP', 'Status', 'HttpVersion', 'Timestamp', 'x0_/', 'x0_//home.php',
       'x0_/action.php', 'x0_/adminpanel.php', 'x0_/allsubmission.php',
       'x0_/announcement.php', 'x0_/archive.php',
       'x0_/bootstrap-3.3.7/js/bootstrap.js',
       'x0_/bootstrap-3.3.7/js/bootstrap.min.js', 'x0_/compile.php',
       'x0_/compiler.php', 'x0_/contest.php', 'x0_/contestproblem.php',
       'x0_/contestshowcode.php', 'x0_/contestsubmission.php',
       'x0_/contestsubmit.php', 'x0_/countdown.php', 'x0_/createadmin.php',
       'x0_/css/bootstrap.min.css', 'x0_/css/bootstrap.min.css.map',
       'x0_/css/font-awesome.min.css', 'x0_/css/main.css',
       'x0_/css/normalize.css', 'x0_/css/style.css',
       'x0_/dboot/js/bootstrap.min.js',
       'x0_/dcss/bootstrap-datetimepicker.min.css', 'x0_/description.php',
       'x0_/details.php', 'x0_/djquery/jquery-1.8.3.min.js',
       'x0_/djs/bootstrap-datetimepicker.js',
       'x0_/djs/vendor/bootstrap-datetimepicker.js', 'x0_/edit.php',
       'x0_/editcontest.php', 'x0_/editcontestproblem.php', 'x0_/favicon.ico',
       'x0_/fonts/fontawesome-webfont.eot',
       'x0_/fonts/fontawesome-webfont.woff',
       'x0_/fonts/fontawesome-webfont.woff2',
       'x0_/fonts/glyphicons-halflings-regular.woff',
       'x0_/fonts/glyphicons-halflings-regular.woff2', 'x0_/home.php',
       'x0_/img/ruet.png', 'x0_/index.php', 'x0_/js/chart.min.js',
       'x0_/js/jquery.min.js', 'x0_/js/vendor/jquery-1.12.0.min.js',
       'x0_/js/vendor/modernizr-2.8.3.min.js', 'x0_/js/vendor/moment.min.js',
       'x0_/login.php', 'x0_/logout.php', 'x0_/pcompile.php',
       'x0_/process.php', 'x0_/profile.php', 'x0_/robots.txt',
       'x0_/setcontest.php', 'x0_/setcontestproblem.php', 'x0_/setproblem.php',
       'x0_/showcode.php', 'x0_/sign.php', 'x0_/standings.php',
       'x0_/submit.php', 'x0_/update.php'],
      dtype='object')
```

Skaalatuilla muuttujilla voidaan sitten harjoittaa mallia ja testata sitä.

In [229]:
```
X_train, X_test = train_test_split(std_df, test_size=0.25)
detector_large = LocalOutlierFactor(novelty=True, contamination="auto")
detector_large.fit(X_train)
```

Out[229]:
```
LocalOutlierFactor(algorithm='auto', contamination='auto', leaf_size=30,
          metric='minkowski', metric_params=None, n_jobs=None,
          n_neighbors=20, novelty=True, p=2)
```

In [244]:
```
detections = detector_large.predict(X_test.iloc[0].values.reshape([1,-1]))
```

Näin monella muuttujalla sekä treenaamisessa, että testaamisessa kului huomattavan paljon enemmän aikaa. Testaaminen on kuitenkin suhteellisen nopeaa, mikäli alkioita ei ole kerralla liian montaa. Sen takia tätä järjestelmää voisi jopa hyödyntää jonkinlaisessa automatisoidussa järjestelmässä, joka luokittelee lokidataa.

Hyödylliset linkit

- Kmodesin lähdekoodi ja github-sivu (https://github.com/nicodv/kmodes)
- Sklearnin dokumentaatio (https://scikit-learn.org/stable/modules/classes.html)
- Holoviewsin dokumentaatio (http://holoviews.org/reference/elements/matplotlib/Scatter3D.html)

Helpot ja hankalat asiat

- Koin holoviewsin erittäin hankalaksi yrittäessäni saada eri värejä eri ryhmille. Enpä tajunnut, että se käy niinkin helposti kuin vain laittamalla *-merkki kyseisten ryhmien välille :D
- K-Prototypes oli suhteellisen näppärä, mutta se osoittautui hyödyttömäksi yli 3 klusterilla. Ehkäpä jokin bugi kirjastossa.

In [251]:
```
# Tämä on seuraavaa steppiä varten
import pickle
with open('detector.model', 'wb') as f:
    pickle.dump(detector_large, f, pickle.HIGHEST_PROTOCOL)
with open('scaler.pickle', 'wb') as f:
    pickle.dump(scaler, f, pickle.HIGHEST_PROTOCOL)
with open('encoder.pickle', 'wb') as f:
    pickle.dump(enc, f, pickle.HIGHEST_PROTOCOL)
```

In [ ]:
```

```