Machine Learning Project

Group members:

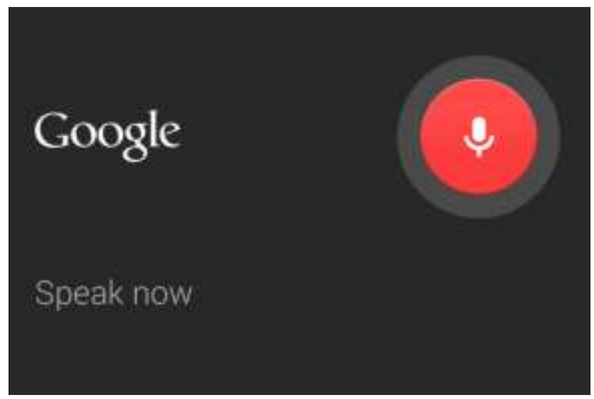
Sidra (EL-015)

Aqsa Sarfaraz(EL-011)

Voice recognizer identify human voice and predict baby male and female voices

Backstory

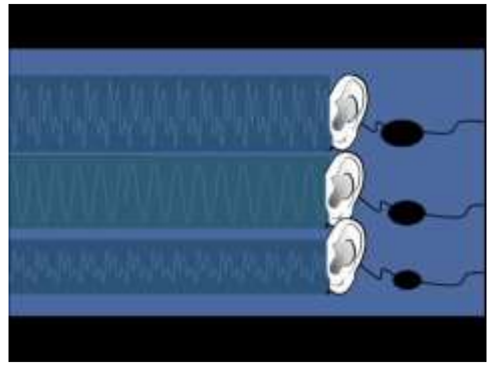
This branch spreads very fast and now it is almost everywhere – military, education, telephony, in-car systems, security or usual life. The simplest example is our smartphones, almost all smartphones support such thing as speech recognition. I wouldn't say that it is the most important thing and that it is necessary for everyone, I would rather say that it takes place in future technologies, such as full-value AI and security. But this technology is simple only in words. Actually it is difficult enough.



Aim

Generally, speech recognition is wide branch and difficult on the one hand but although very interesting on the another hand. So in this presentation I'd like to show the way we can recognize gender of person's voice.

Dataset



Make a long story short sound is a pressure wave which is created by a vibrating object. Humans can hear sound waves with frequencies between about 20 Hz and 20 kHz. Analyzed frequency range of 0hz-280hz.

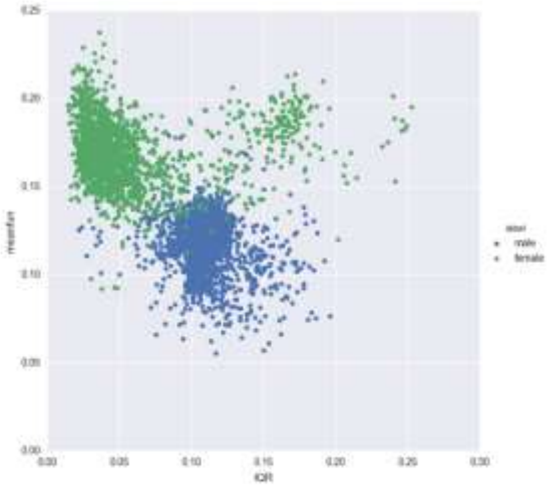
The dataset consists of 3168 recorded voice samples, collected from (baby) male and female speakers.

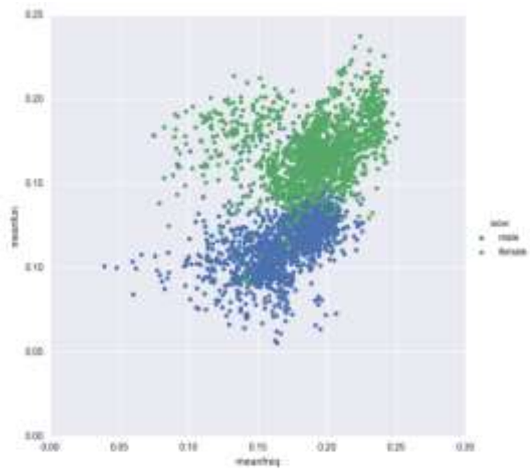
The following acoustic properties of each voice:

- **meanfreq**: mean frequency (in kHz)
- **sd**: standard deviation of frequency
- **median**: median frequency (in kHz)
- **Q25**: first quantile (in kHz)
- **Q75**: third quantile (in kHz)
- **IQR**: interquantile range (in kHz)
- **skew**: skewness (see note in specprop description)
- **kurt**: kurtosis (see note in specprop description)
- **sp.ent**: spectral entropy
- **sfm**: spectral flatness
- **mode**: mode frequency
- **centroid**: frequency centroid (see specprop)
- **peakf**: peak frequency (frequency with highest energy)
- **meanfun**: average of fundamental frequency measured across acoustic signal
- **minfun**: minimum fundamental frequency measured across acoustic signal
- **maxfun**: maximum fundamental frequency measured across acoustic signal
- **meandom**: average of dominant frequency measured across acoustic signal
- **mindom**: minimum of dominant frequency measured across acoustic signal
- **maxdom**: maximum of dominant frequency measured across acoustic signal

- **dfrange**: range of dominant frequency measured across acoustic signal
- **modindx**: modulation index. Calculated as the accumulated absolute difference between adjacent measurements of fundamental frequencies divided by the frequency range
- **label**: (baby) male or female

The most significant features: meanfun and IQR

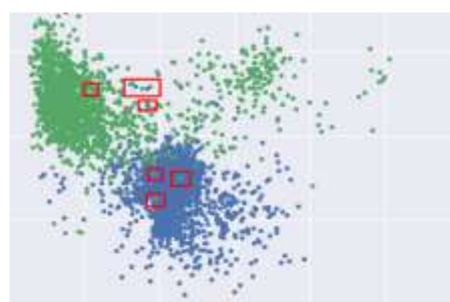




From all 21 features we can separate “meanfun” and “IQR”. Is it enough of these two features to make predictions? We will see later.

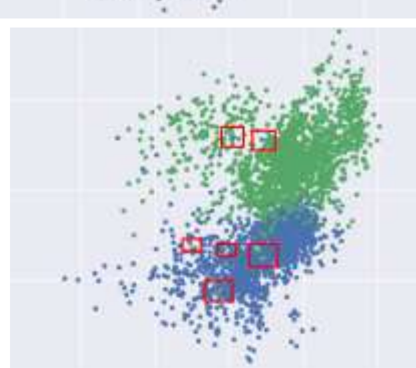
It is possible because of next reasons:

- particular qualities
- accuracy of equipment not enough
- interference during recording
- speech defects



As we could see, there are some samples

that belong to male but graph tells us that it is female. And vice versa.



Prediction results

For making prediction I've used 4 algorithms: Logistic Regression, K-means, Naive Bayes and SVM. Also, I found that feature scaling improves results, except K-means.

Below you can see results of each algorithm with 2534 train samples and 634 test samples with all features and with 2 features separately.

Features	Feature scaling	K-means	Logistic Regression
All	Without	54.416 % clusters: 2	88.17 %
All	With	53.31 % clusters: 1	97.003 %
'meanfun', 'IQR'	Without	88.96 % clusters: 2	90.22 %
'meanfun', 'IQR'	With	53.31 % clusters: 1	95.58 %

CODING AND OUTPUTS

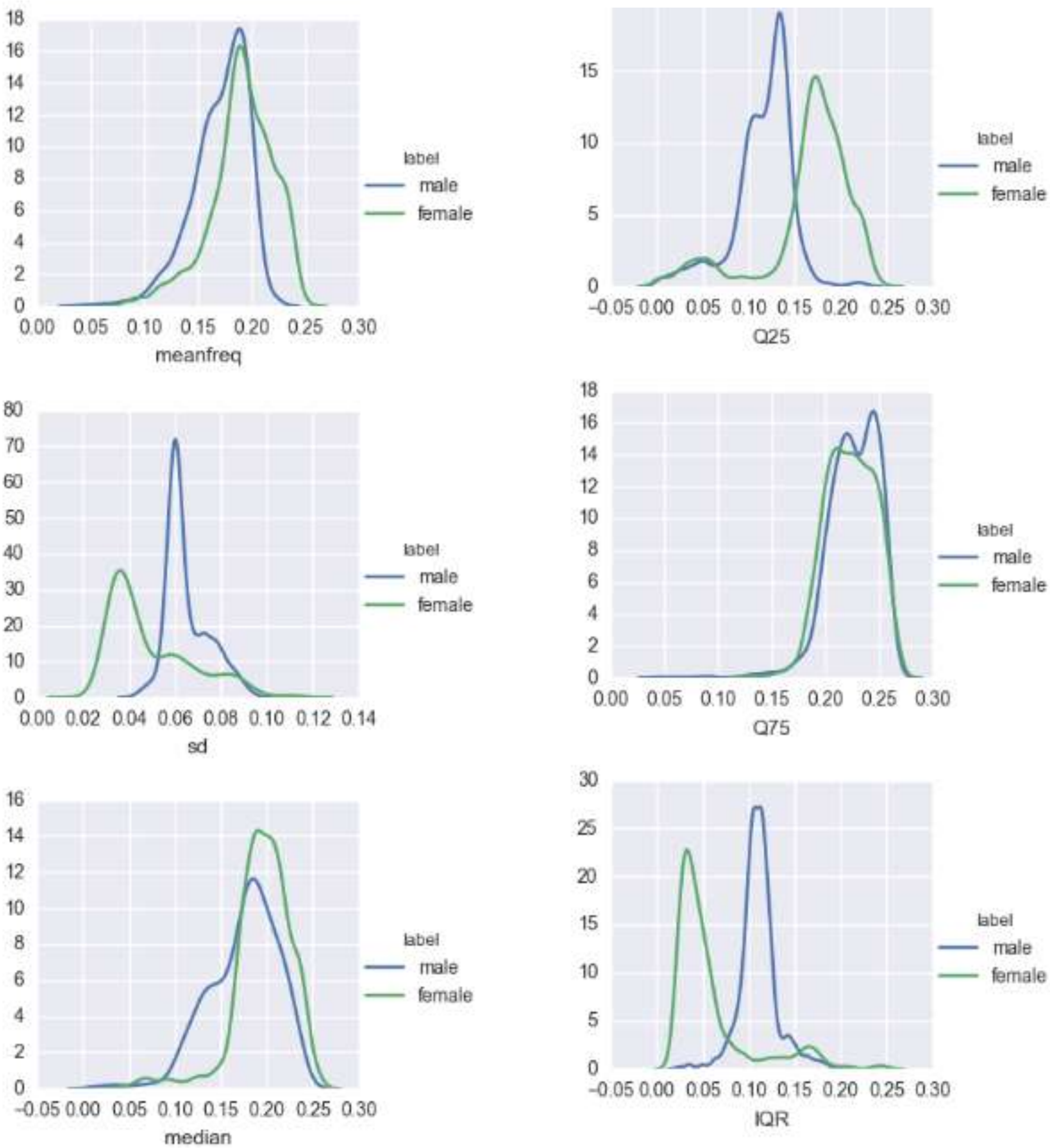
```
import pandas as pd

voice = pd.read_csv('voice.csv')
print(voice.head())
voice["label"].value_counts()
```

```
   meanfreq        sd    median       Q25  ...    maxdom   dfrange   modindx  label
0  0.059781  0.064241  0.032027  0.015071  ...  0.007812  0.000000  0.000000   male
1  0.066009  0.067310  0.040229  0.019414  ...  0.054688  0.046875  0.052632   male
2  0.077316  0.083829  0.036718  0.008701  ...  0.015625  0.007812  0.046512   male
3  0.151228  0.072111  0.158011  0.096582  ...  0.562500  0.554688  0.247119   male
4  0.135120  0.079146  0.124656  0.078720  ...  5.484375  5.476562  0.208274   male

[5 rows x 21 columns]
Out[12]:
male      1584
female    1584
Name: label, dtype: int64
```

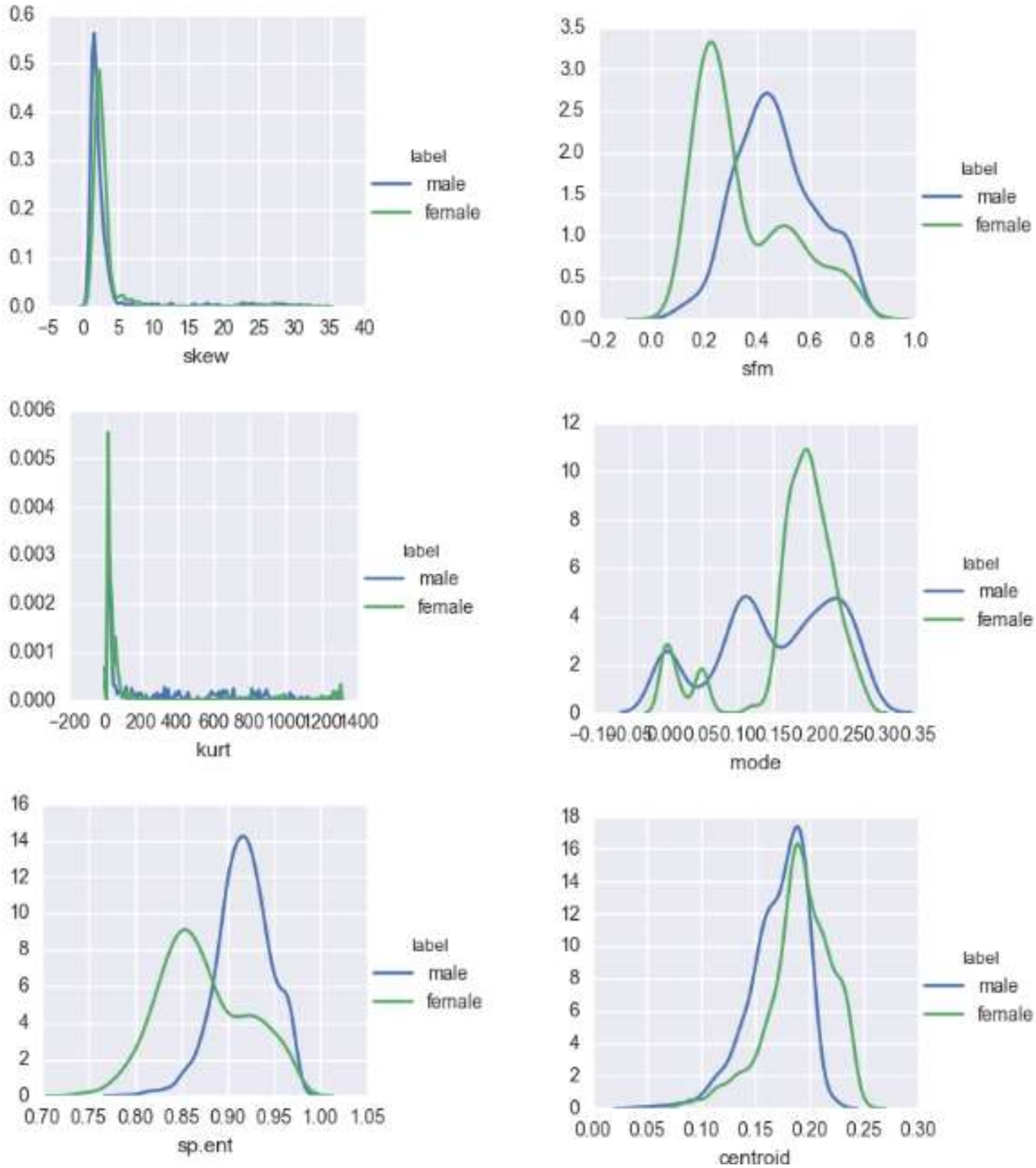
```
voice.info()
```

```
In [13]: voice.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
meanfreq    3168 non-null float64
sd          3168 non-null float64
median      3168 non-null float64
Q25         3168 non-null float64
Q75         3168 non-null float64
IQR         3168 non-null float64
skew        3168 non-null float64
kurt        3168 non-null float64
sp.ent      3168 non-null float64
sfm         3168 non-null float64
mode        3168 non-null float64
centroid    3168 non-null float64
meanfun     3168 non-null float64
minfun      3168 non-null float64
maxfun      3168 non-null float64
meandom     3168 non-null float64
mindom      3168 non-null float64
maxdom      3168 non-null float64
dfrange     3168 non-null float64
modindx     3168 non-null float64
label       3168 non-null object
dtypes: float64(20), object(1)
memory usage: 519.9+ KB
```

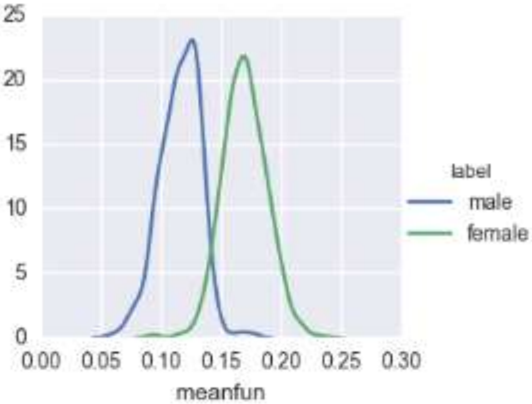
```
import seaborn as sns
from seaborn import plt

for col in voice.columns[:-1]:
    sns.FacetGrid(voice, hue="label", size=3).map(sns.kdeplot, col).add_le
gend()
    plt.show()
```

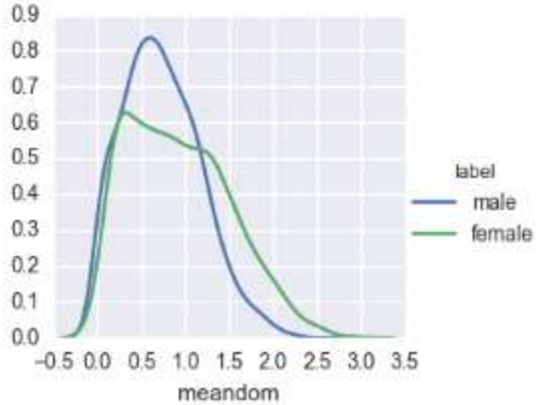


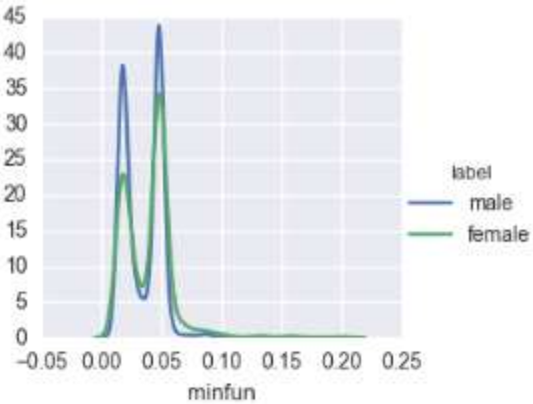
meanfreq
Q25
sd
Q75
median
IQR
label
male
female

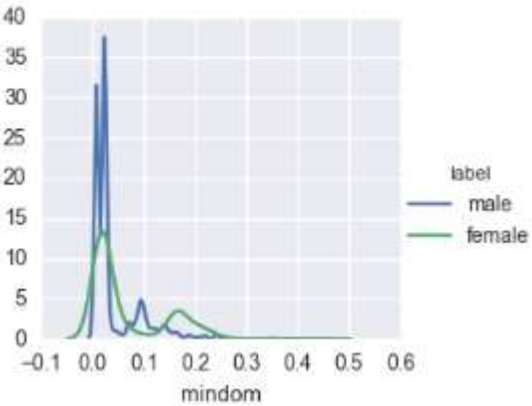


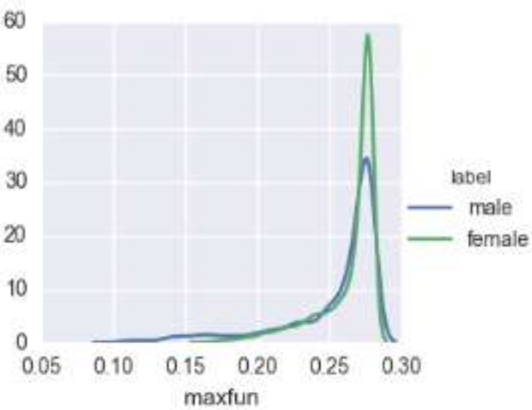
skew
label
male
female
sfm
label
male
female
kurt
label
male
female
mode
label
male
female
sp.ent
label
male
female
centroid
label
male
female

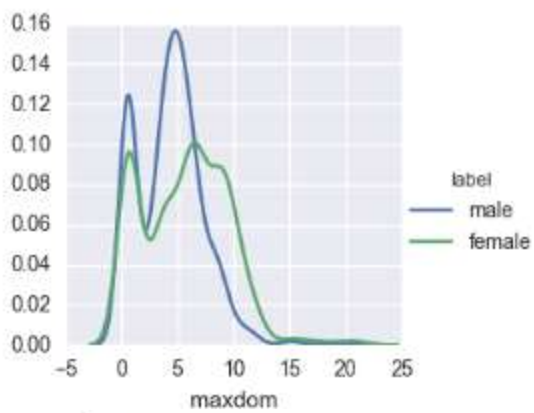

0 5 10 15 20 25
0.00 0.05 0.10 0.15 0.20 0.25 0.30
meanfun
label
male
female

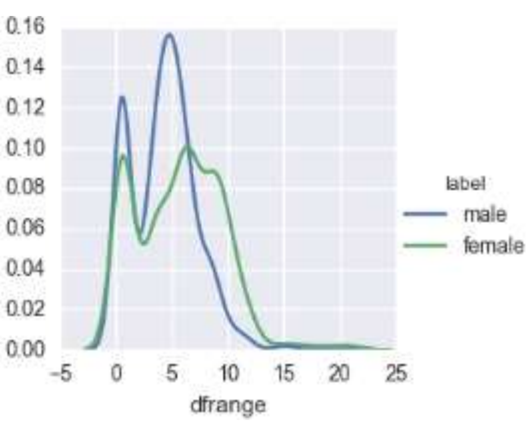

0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
−0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5
meandom
label
male
female

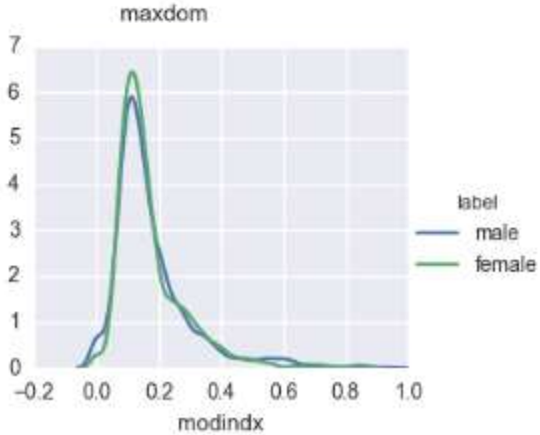

0 5 10 15 20 25 30 35 40 45
−0.05 0.00 0.05 0.10 0.15 0.20 0.25
minfun
label
male
female


0 5 10 15 20 25 30 35 40
−0.1 0.0 0.1 0.2 0.3 0.4 0.5 0.6
mindom
label
male
female

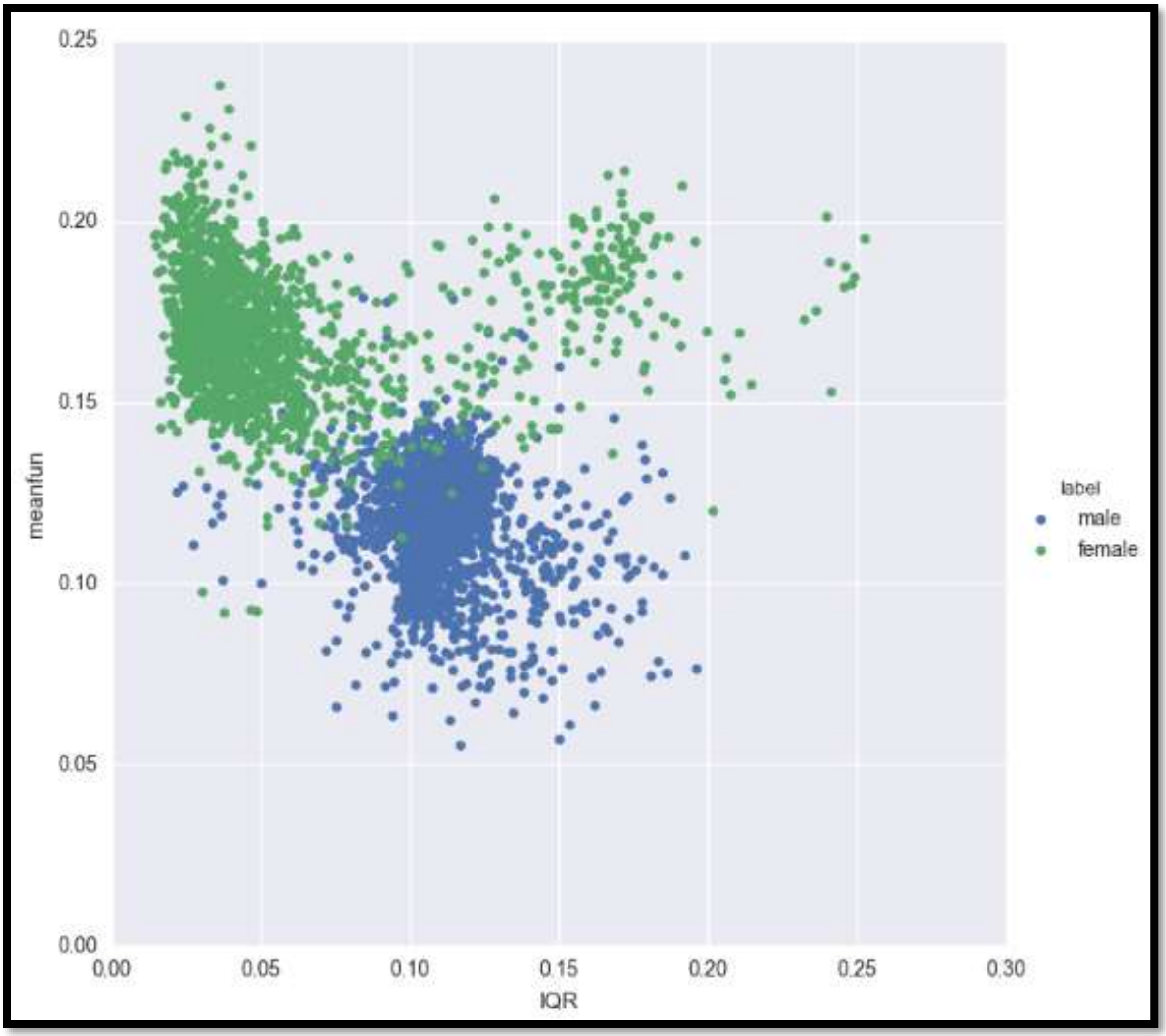

0 10 20 30 40 50 60
0.05 0.10 0.15 0.20 0.25 0.30
maxfun
label
male
female


0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16
−5 0 5 10 15 20 25
maxdom
label
male
female

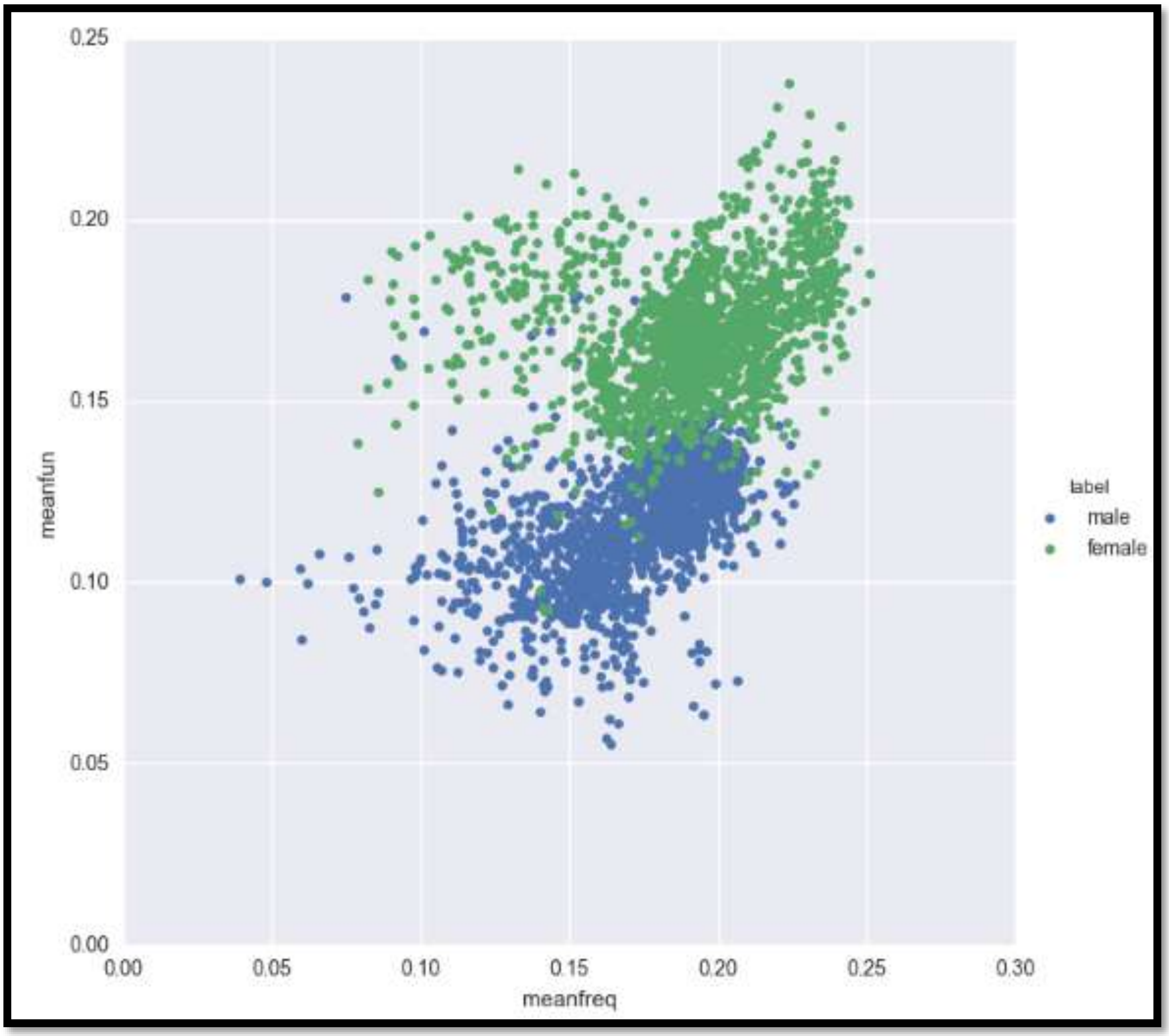

0.00 0.02 0.04 0.06 0.08 0.10 0.12 0.14 0.16
−5 0 5 10 15 20 25
dfrange
label
male
female


0 1 2 3 4 5 6 7
−0.2 0.0 0.2 0.4 0.6 0.8 1.0
modindx
label
male
female

```
sns.FacetGrid(voice, hue="label", size=7).map(plt.scatter, "IQR", "meanfun
").add_legend()
plt.show()
```



```
sns.FacetGrid(voice, hue="label", size=7).map(plt.scatter, "meanfreq", "meanf
un").add_legend()
plt.show()
```




```
from sklearn.preprocessing import LabelEncoder

# replace male/female => 1/0
gender_encoder = LabelEncoder()
voice['label'] = gender_encoder.fit_transform(voice.iloc[:, -1])
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cross_validation import train_test_split

y=voice['label']
x = voice.iloc[:, :-1]
scaler = StandardScaler()
scaler.fit(x)
x = scaler.transform(x)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=2)

# code without scale below

#train, test = train_test_split(voice, test_size=0.2, random_state=2)

# separating data in features and labels
#x_train = train.iloc[:, :-1]
#y_train = train.iloc[:, -1]
#x_test = test.iloc[:, :-1]
#y_test = test.iloc[:, -1]

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

lr = LogisticRegression()
lr.fit(x_train, y_train)
prediction = lr.predict(x_test)

print('LR result: ', accuracy_score(y_test, prediction))
```

```
LR result:  0.970031545741
```

```
from sklearn.cluster import KMeans
import math

result = 0
for num_clusters in range(1, len(x_train)):
    kmeans = KMeans(n_clusters=num_clusters, random_state=0)
    kmeans.fit(x_train, y_train)
    prediction = kmeans.predict(x_test)
    cur_result = accuracy_score(y_test, prediction)
    if cur_result < result or math.fabs(cur_result-result) < 0.01:
        break
    result = cur_result

print('K-means result: ', result, '. Cluster quantity = ', num_clusters-1)
```

```
K-means result:  0.533123028391 . Cluster quantity =  1
```

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
prediction = nb.predict(x_test)

print('NB result: ', accuracy_score(y_test, prediction))
```

```
NB result:  0.862776025237
```

```
from sklearn.svm import SVC

svc=SVC()
svc.fit(x_train,y_train)
prediction = svc.predict(x_test)
print('SVM result: ', accuracy_score(y_test,prediction))
```

```
SVM result:  0.97476340694
```

```
voice_cut = voice[['IQR', 'meanfun', 'label']]

#y_cut=voice['label']
#x = voice_cut.iloc[:, :-1]
#scaler = StandardScaler()
#scaler.fit(x)
#x_cut = scaler.transform(x)

#x_train, x_test, y_train, y_test = train_test_split(x_cut, y_cut, test_si
ze=0.2, random_state=2)

train, test = train_test_split(voice_cut, test_size=0.2, random_state=2)

# separating data in features and labels
x_train = train.iloc[:, :-1]
y_train = train.iloc[:, -1]
x_test = test.iloc[:, :-1]
y_test = test.iloc[:, -1]


lr = LogisticRegression()
lr.fit(x_train, y_train)
prediction = lr.predict(x_test)

print('LR result with 2 features: ', accuracy_score(y_test, prediction))
```

```
LR result with 2 features:  0.902208201893
```

```
result = 0
for num_clusters in range(1, len(x_train)):
    kmeans = KMeans(n_clusters=num_clusters, random_state=0)
    kmeans.fit(x_train, y_train)
    prediction = kmeans.predict(x_test)
    cur_result = accuracy_score(y_test, prediction)
    if cur_result < result or math.fabs(cur_result-result) < 0.01:
        break
```

```
    result = cur_result

print('K-means result with 2 features: ', result, '. Cluster quantity = ', num_clusters-1)
```

```
K-means result with 2 features:  0.889589905363 . Cluster quantity =  2
```

```
nb = GaussianNB()
nb.fit(x_train, y_train)
prediction = nb.predict(x_test)

print('NB result with 2 features: ', accuracy_score(y_test, prediction))
```

```
NB result with 2 features:  0.958990536278
```

```
svc=SVC()
svc.fit(x_train,y_train)
prediction = svc.predict(x_test)

print('SVM result with 2 features: ', accuracy_score(y_test,prediction))
```

```
SVM result with 2 features:  0.900630914826
```

Conclusion

So, best result results we have in SVM, then logistic regression, Naive Bayes and K-means.

But in case with only 2 features(‘meanfun’, ‘IQR’) without feature scaling results are more precise. Naive Bayers, for example, gives 95 % against 85-85% with all features. That is because algorithm is based on suggestion of independence. The most significant improvement is in K-means without feature scaling, it grows up from ~54% to 88%. The reason is the same, less quantity of features.

Generally, results of partial dataset don’t lose to full, but sometimes gives even better results. Hence, in some cases we can use only 2 features, instead of all 21.