AUTO-FEATURE SELECTION & PARAMETER TUNING

CSE 599 Prototyping Interactive Systems | Lecture 18 | May 30

Jon Froehlich • Jasper O'Leary (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Python Data Analysis Library

pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

pandas is a NumFOCUS sponsored project. This will help ensure the success of development of *pandas* as a world-class open-source project, and makes it possible to donate to the project.

A Fiscally Sponsored Project of

NUMFOCUS

OPEN CODE = BETTER SCIENCE

VERSIONS

Release

0.24.2 - March 2019

download // docs // pdf

Development

0.25.0 - April 2019

github // docs

Previous Releases

0.24.1 - download // docs // pdf

0.24.0 - download // docs // pdf

0.23.4 - download // docs // pdf

0.23.3 - download // docs // pdf

pandas.DataFrame

class `pandas.`**`DataFrame`**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*) [source]

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Parameters:

data : *ndarray (structured or homogeneous), Iterable, dict, or DataFrame*
Dict can contain Series, arrays, constants, or list-like objects
Changed in version 0.23.0: If data is a dict, argument order is maintained for Python 3.6 and later.

index : *Index or array-like*
Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided

columns : *Index or array-like*
Column labels to use for resulting frame. Will default to RangeIndex (0, 1, 2, …, n) if no column labels are provided

dtype : *dtype, default None*
Data type to force. Only a single dtype is allowed. If None, infer

copy : *boolean, default False*
Copy data from inputs. Only affects DataFrame / 2d ndarray input

See also:

`DataFrame.from_records`
Constructor from tuples, also record arrays.

`DataFrame.from_dict`
From dicts of Series, arrays, or dicts.

`DataFrame.from_items`

	gesturer	gesture	trial_num	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag.mean	x.mean	...	mag_p.top3_freq	mag_p.top3_fr
0	JonGesturesEasy	At Rest	0	349	-0.001228	0.000537	-0.003691	0.000840	941.461182	487.939828	...	2.859028	0
1	JonGesturesEasy	At Rest	1	329	0.002189	0.001214	0.001711	0.000938	941.442396	487.802432	...	3.266449	0
2	JonGesturesEasy	At Rest	2	266	0.003965	-0.003338	0.000780	0.008070	941.570441	487.887218	...	2.599653	0
3	JonGesturesEasy	At Rest	3	411	-0.001210	0.000694	0.000418	-0.002770	941.647837	487.922141	...	1.867065	0
4	JonGesturesEasy	At Rest	4	443	0.003060	-0.000471	0.003076	0.002459	941.695043	487.954853	...	1.904432	0
5	JonGesturesEasy	Backhand Tennis	0	224	0.412261	0.476257	0.009189	0.112804	978.420645	542.017857	...	1.029160	25
6	JonGesturesEasy	Backhand Tennis	1	219	0.317739	0.343668	0.082957	0.055485	974.979163	535.164384	...	1.404494	18
7	JonGesturesEasy	Backhand Tennis	2	234	0.344929	0.424049	0.058713	0.035865	974.949140	535.525641	...	0.984898	19
8	JonGesturesEasy	Backhand Tennis	3	203	0.437270	0.445153	0.187909	0.032496	983.168663	545.261084	...	1.515726	23
9	JonGesturesEasy	Backhand Tennis	4	254	0.255910	0.300459	0.039217	0.046184	976.387843	537.562992	...	1.512402	17
10	JonGesturesEasy	Baseball Throw	0	280	0.238396	0.327967	0.180251	-0.125919	971.279581	517.967857	...	0.274273	12
11	JonGesturesEasy	Baseball Throw	1	306	0.176786	0.209537	0.182245	-0.138652	980.382968	525.993464	...	0.501756	12
12	JonGesturesEasy	Baseball Throw	2	333	0.141593	0.180343	0.085029	-0.075302	994.310399	548.222222	...	0.461148	15
13	JonGesturesEasy	Baseball Throw	3	331	0.165884	0.179856	0.165543	-0.102396	989.807322	544.619335	...	0.231803	18
14	JonGesturesEasy	Baseball Throw	4	282	0.319356	0.422032	0.255844	-0.205548	1008.424162	567.460993	...	0.544662	21

```
import pandas as pd

gesture_sets = get_all_gesture_sets()
(list_of_feature_vectors, feature_names) = extract_features_from_gesture_sets(gesture_sets)
df = pd.DataFrame(list_of_feature_vectors, columns = feature_names)
display(df)
```

	gesturer	gesture	trial_num	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag.mean	x.mean	...	mag_p.top3_freq	mag_p.top3_fr
0	JonGesturesEasy	At Rest	0	349	-0.001228	0.000537	-0.003691	0.000840	941.461182	487.939828	...	2.859028	0
1	JonGesturesEasy	At Rest	1	329	0.002189	0.001214	0.001711	0.000938	941.442396	487.802432	...	3.266449	0
2	JonGesturesEasy	At Rest	2	266	0.003965	-0.003338	0.000780	0.008070	941.570441	487.887218	...	2.599653	0
3													0
4													0
5													25
6													18
7	JonGesturesEasy	Backhand Tennis	2	234	0.344929	0.424049	0.058713	0.035865	974.949140	535.525641	...	0.984898	19
8	JonGesturesEasy	Backhand Tennis	3	203	0.437270	0.445153	0.187909	0.032496	983.168663	545.261084	...	1.515726	23
9	JonGesturesEasy	Backhand Tennis	4	254	0.255910	0.300459	0.039217	0.046184	976.387843	537.562992	...	1.512402	17
10	JonGesturesEasy	Baseball Throw	0	280	0.238396	0.327967	0.180251	-0.125919	971.279581	517.967857	...	0.274273	12
11	JonGesturesEasy	Baseball Throw	1	306	0.176786	0.209537	0.182245	-0.138652	980.382968	525.993464	...	0.501756	12
12	JonGesturesEasy	Baseball Throw	2	333	0.141593	0.180343	0.085029	-0.075302	994.310399	548.222222	...	0.461148	15
13	JonGesturesEasy	Baseball Throw	3	331	0.165884	0.179856	0.165543	-0.102396	989.807322	544.619335	...	0.231803	18
14	JonGesturesEasy	Baseball Throw	4	282	0.319356	0.422032	0.255844	-0.205548	1008.424162	567.460993	...	0.544662	21

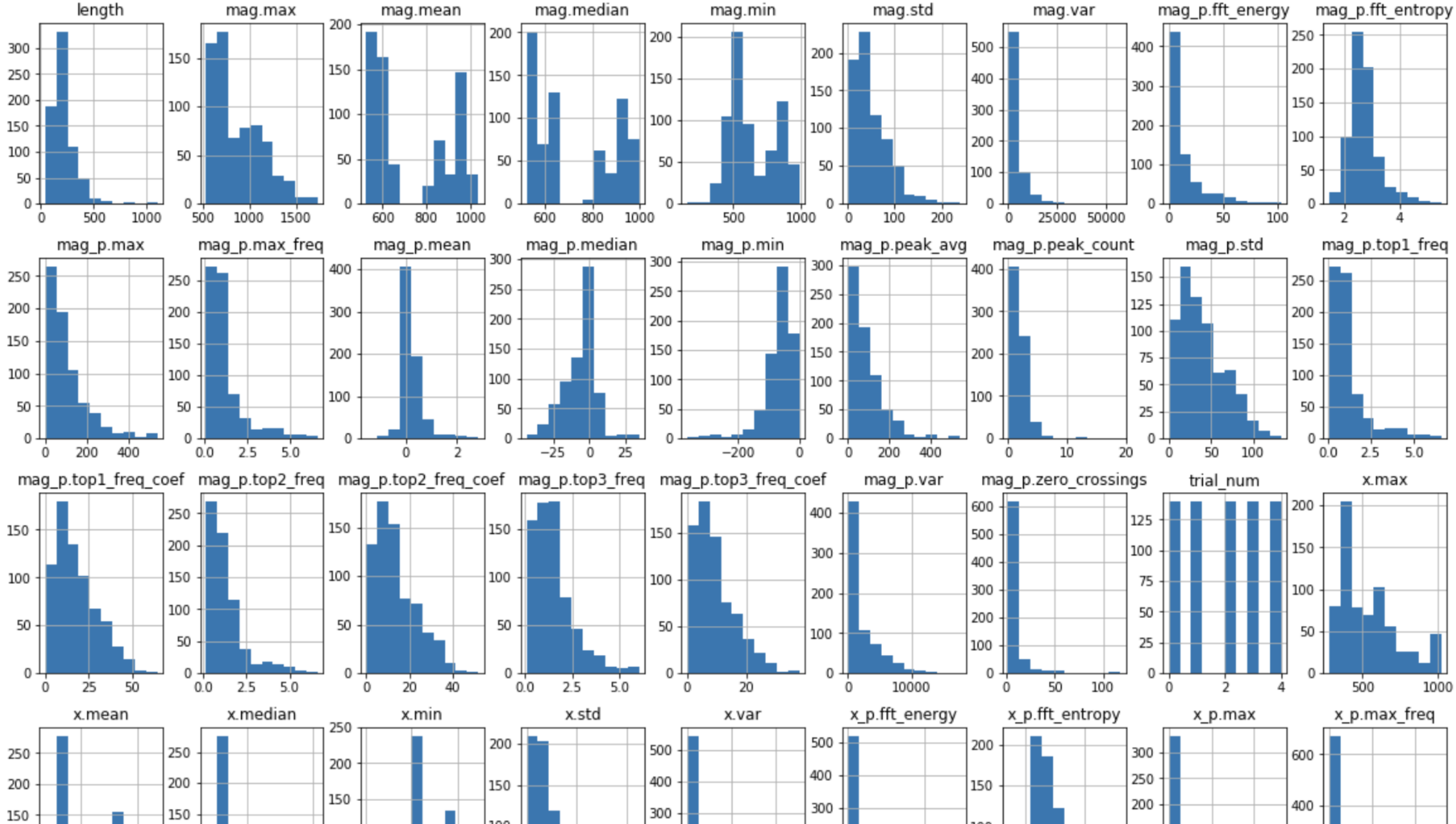
PANDAS

DATAFRAME DESCRIBE()

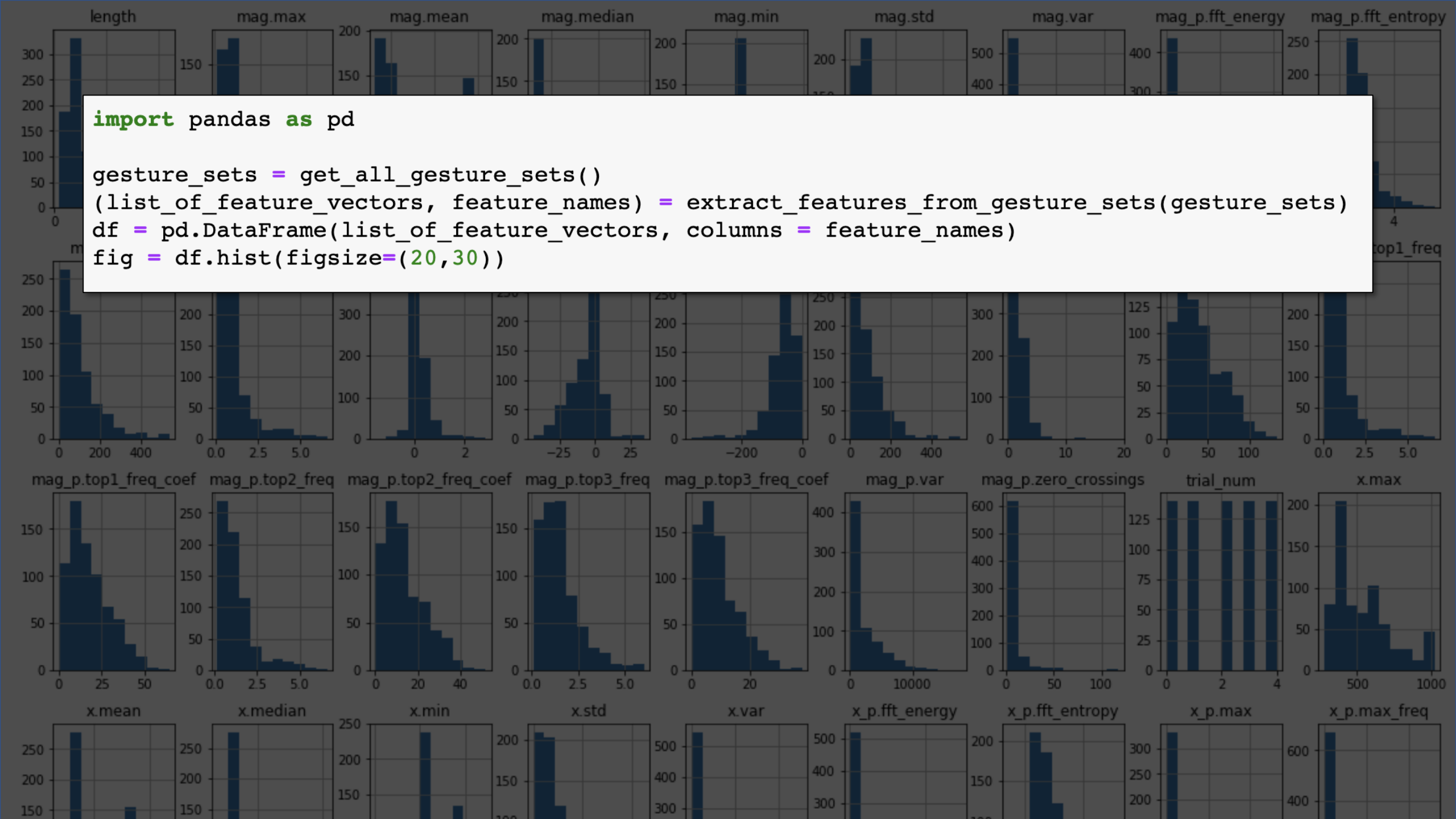
In [20]:
```
import pandas as pd

gesture_sets = get_all_gesture_sets()
(list_of_feature_vectors, feature_names) = extract_features_from_gesture_sets(gesture_sets)
df = pd.DataFrame(list_of_feature_vectors, columns = feature_names)
display(df.describe())
```

	trial_num	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag.mean	x.mean	y.mean	z.mean	...	mag_p.top3_freq	mag_
count	770.000000	770.000000	770.000000	770.000000	770.000000	770.000000	770.000000	770.000000	770.000000	770.000000	...	770.000000	
mean	2.000000	219.664935	0.209935	0.254837	-0.020124	0.022725	728.971642	415.100544	404.732558	429.912676	...	1.517669	
std	1.415133	117.652922	0.441457	0.513010	0.203509	0.348028	174.923875	99.258016	103.870000	115.533689	...	1.037399	
min	0.000000	40.000000	-1.567288	-1.696831	-0.855175	-2.420936	525.019769	264.743902	254.602740	272.612069	...	0.097618	
25%	1.000000	145.250000	-0.004146	0.000570	-0.091245	-0.054498	567.398727	329.921863	322.210291	320.752321	...	0.751107	
50%	2.000000	190.000000	0.097635	0.166246	-0.001402	0.001192	626.351592	353.968096	344.011956	408.191217	...	1.345292	
75%	3.000000	260.000000	0.351606	0.402773	0.071658	0.075966	935.597491	515.953104	509.723483	526.282477	...	1.954397	
max	4.000000	1100.000000	2.862590	3.652698	1.116560	3.280134	1036.170629	667.952830	618.300000	625.700893	...	6.044325	

8 rows × 80 columns


length
mag.max
mag.mean
mag.median
mag.min
mag.std
mag.var
mag_p.fft_energy
mag_p.fft_entropy
mag_p.max
mag_p.max_freq
mag_p.mean
mag_p.median
mag_p.min
mag_p.peak_avg
mag_p.peak_count
mag_p.std
mag_p.top1_freq
mag_p.top1_freq_coef
mag_p.top2_freq
mag_p.top2_freq_coef
mag_p.top3_freq
mag_p.top3_freq_coef
mag_p.var
mag_p.zero_crossings
trial_num
x.max
x.mean
x.median
x.min
x.std
x.var
x_p.fft_energy
x_p.fft_entropy
x_p.max
x_p.max_freq

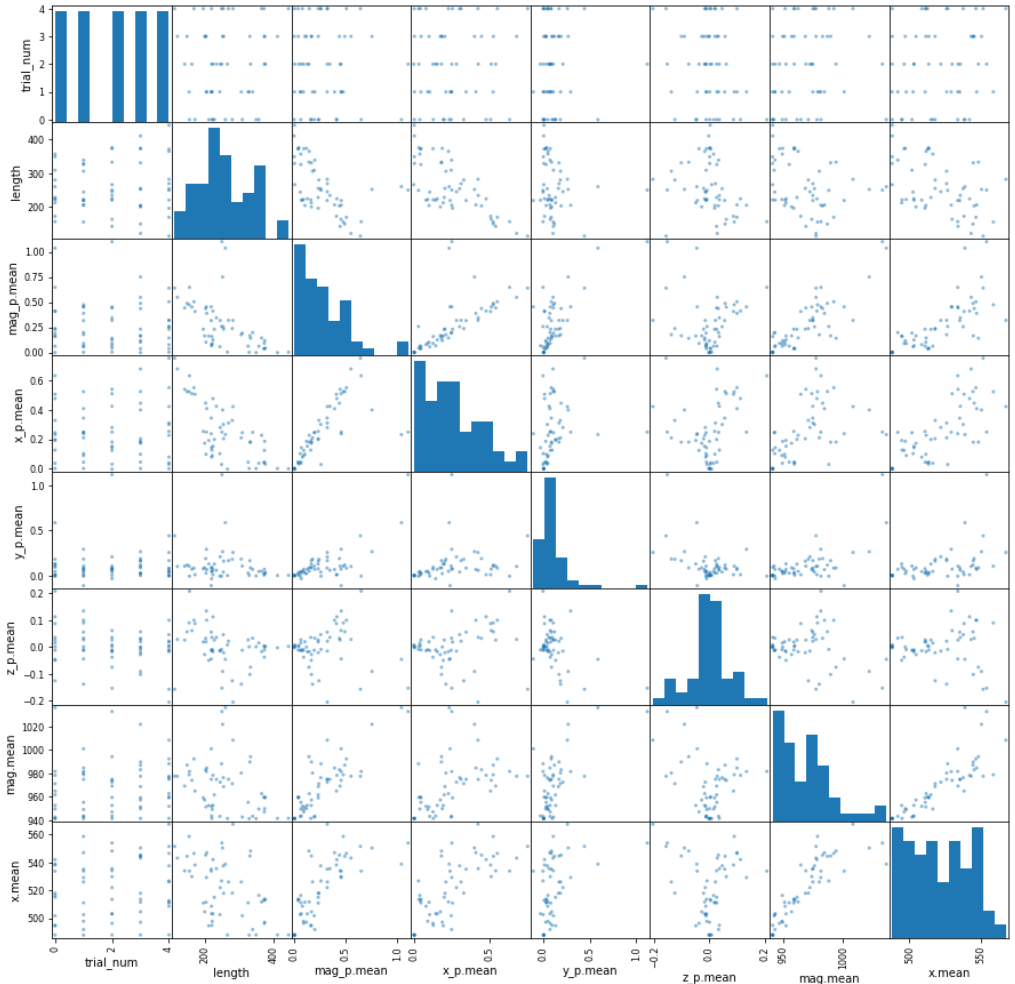
```
import pandas as pd

gesture_sets = get_all_gesture_sets()
(list_of_feature_vectors, feature_names) = extract_features_from_gesture_sets(gesture_sets)
df = pd.DataFrame(list_of_feature_vectors, columns = feature_names)
fig = df.hist(figsize=(20,30))
```



PANDAS FEATURE EXPLORATION

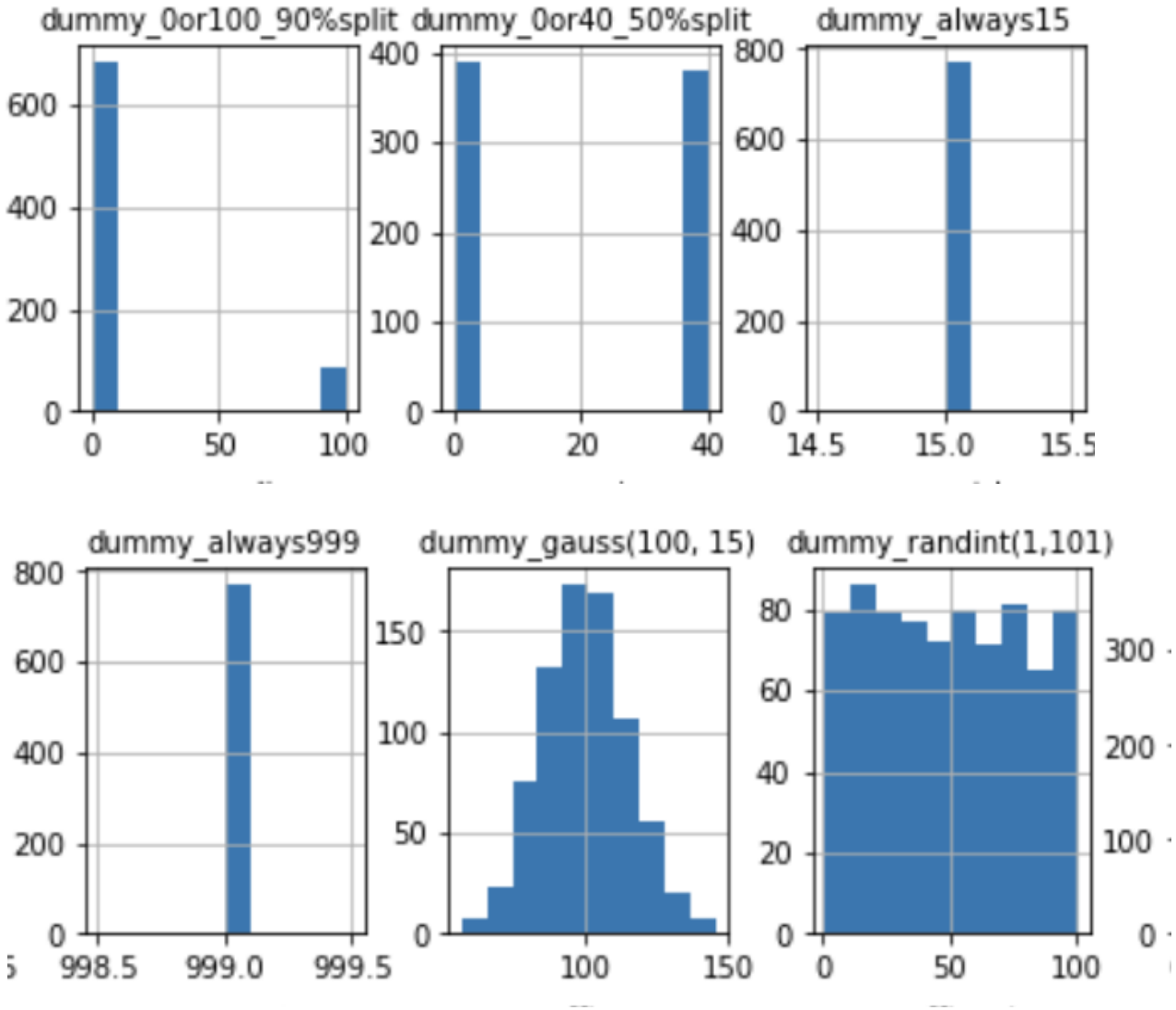
PANDAS SCATTER MATRIX

```
from pandas.plotting import scatter_matrix
scatter_matrix(df)
```



DUMMY/FAKE FEATURES TO TEST FEATURE SELECTION

```
# dummy vars to test feature selection
if include_dummy_data:
    features.append(15)
    feature_names.append("dummy_always15")

    features.append(999)
    feature_names.append("dummy_always999")

    val0or40 = 0
    if random.random() > 0.5:
        val0or40 = 40
    features.append(val0or40)
    feature_names.append("dummy_0or40_50%split")

    val0or100 = 0
    if random.random() > 0.9:
        val0or100 = 100
    features.append(val0or100)
    feature_names.append("dummy_0or100_90%split")

    features.append(random.randint(1,101))
    feature_names.append("dummy_randint(1,101)")

    features.append(random.gauss(100, 15))
    feature_names.append("dummy_gauss(100, 15)")
```



FEATURE SELECTION

PANDAS DATA FRAME

```
import pandas as pd
df = pd.DataFrame(list_of_feature_vectors, columns = column_headers)
display(df)
```

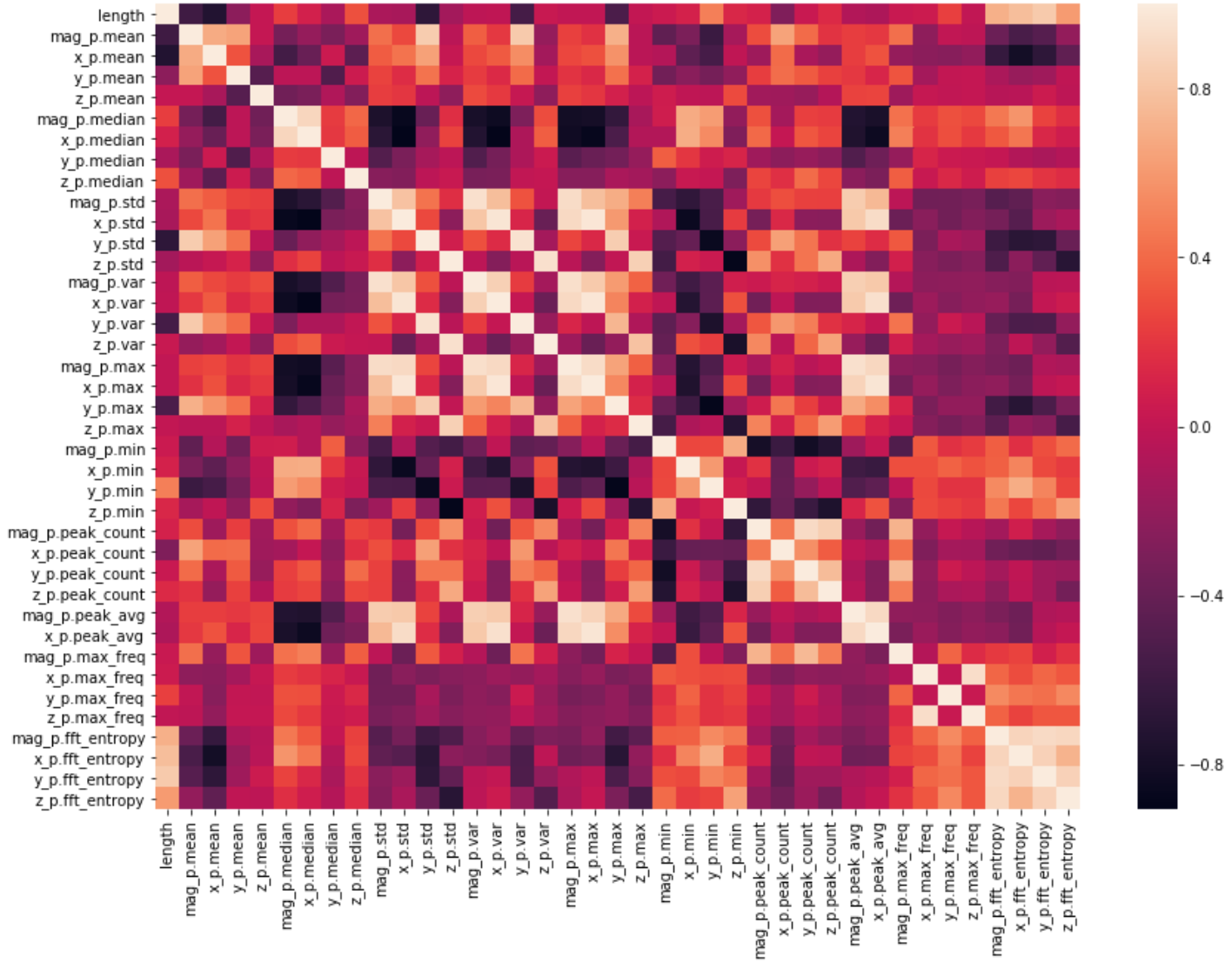
	gesture	trial_num	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag_p.median	x_p.median	y_p.median	z_p.median
0	Midair Zorro 'Z'	0	156	2.486061	2.476347	-0.038095	0.659445	-26.356490	-20.300688	1.924996	-2.978864
1	Midair Zorro 'Z'	1	206	1.530894	1.631314	0.186097	0.381839	-16.724810	-18.826979	0.473243	-6.318860
2	Midair Zorro 'Z'	2	163	1.740777	1.962262	0.349036	0.117441	-20.203598	-15.572067	-2.201870	-0.657275
3	Midair Zorro 'Z'	3	149	1.977666	2.082033	0.452325	0.235048	-18.211218	-14.411052	0.723463	-1.675665
4	Midair Zorro 'Z'	4	169	2.129583	2.199460	0.413543	0.408136	-15.483657	-17.916814	0.075533	-4.376140
5	Baseball Throw	0	280	0.771997	1.154652	0.511057	-0.457802	-10.356645	-8.941720	12.211538	1.529668
6	Baseball Throw	1	306	0.563156	0.672193	0.599782	-0.508517	-11.435248	-9.770039	0.072771	5.889819
7	Baseball Throw	2	333	0.550743	0.633333	0.348090	-0.277344	-14.705581	-21.656489	-4.079071	6.587493
8	Baseball Throw	3	331	0.608548	0.629733	0.616082	-0.392998	-16.626730	-17.376936	-8.597742	2.104689
9	Baseball Throw	4	282	1.331867	1.640106	1.068490	-0.766823	-23.186873	-32.104320	1.474725	0.336937
10	Midair 'S'	0	172	1.630848	1.950385	0.161842	0.245648	-17.581523	-8.367371	-4.051625	-2.612871
11	Midair 'S'	1	155	1.694806	1.901635	0.367123	0.222470	-12.701052	-6.894322	-1.696207	1.719000
12	Midair 'S'	2	142	1.639288	2.120100	0.430372	-0.178143	-8.503183	-3.297189	-0.440282	0.773979
13	Midair 'S'	3	121	1.837432	2.612869	0.244635	-0.275391	-7.338217	7.189629	3.647443	-2.955958
14	Midair 'S'	4	113	1.832756	2.085960	2.028100	2.062896	0.021324	1.864702	1.184038	4.486060

...

FEATURE SELECTION

CORRELATION MATRIX

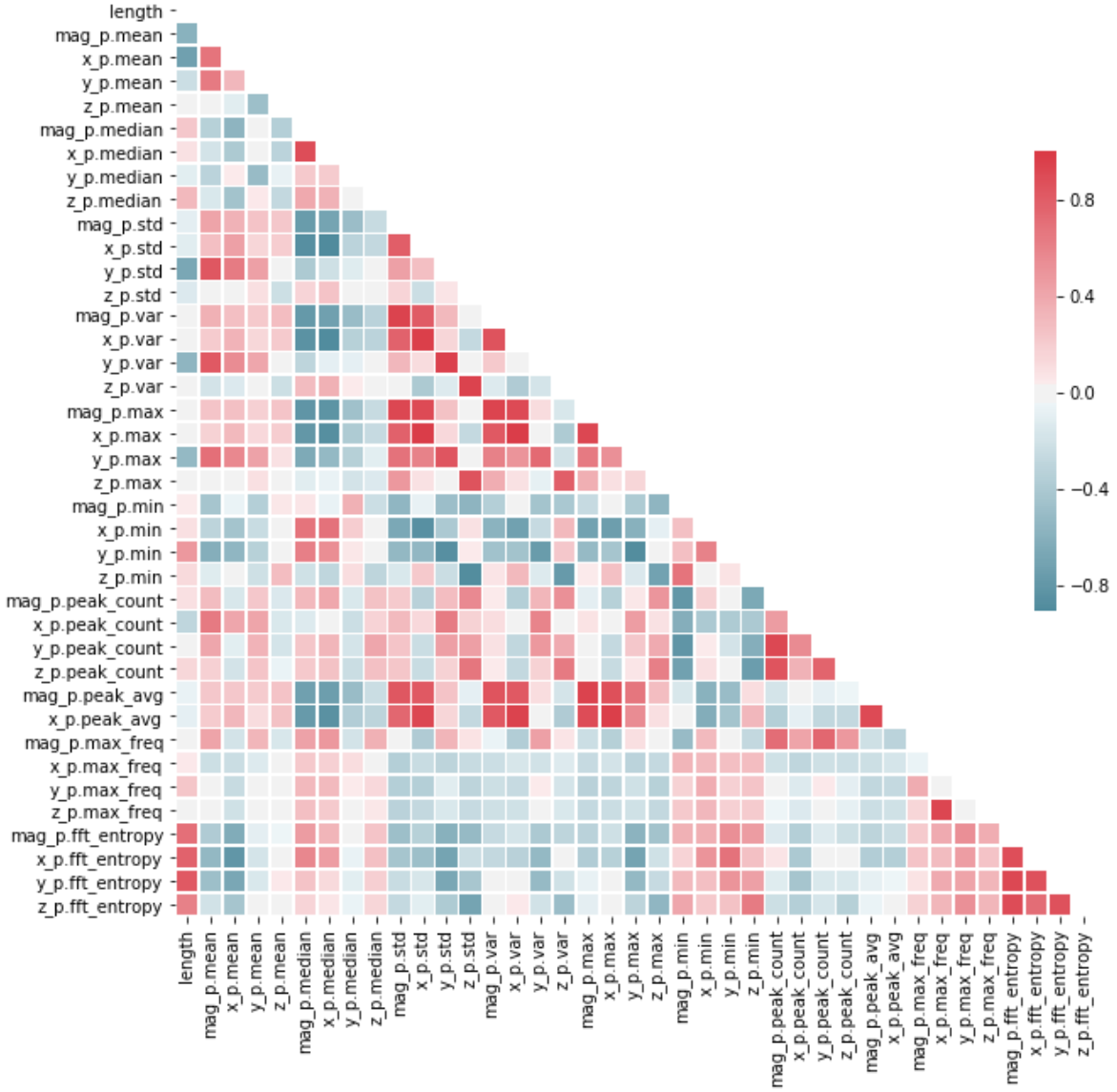
```
import seaborn as sns
plt.figure(figsize=(14,10))
sns.heatmap(df.corr())
```



FEATURE SELECTION

PRETTIFYING CORRELATION MATRIX

```
# Compute the correlation matrix
import seaborn as sns
corr = df.corr()

# Generate a mask for the upper triangle
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(14, 10))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10,
                             as_cmap=True)

# Draw heatmap w/the mask & correct aspect
ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1,
            center=0, square=True,
            linewidths=.5,
            cbar_kws={"shrink": .5})
```



Source: https://seaborn.pydata.org/examples/many_pairwise_correlations.html

Table of Contents

Let's try it **ourselves!**

(please don't share this notebook, it's currently shared in Canvas but not github)

In [1]:
```
# This cell includes the major classes used in our classification analyses
import matplotlib.pyplot as plt # needed for plotting
import numpy as np # numpy is primary library for numeric array (and matrix) handling
import scipy as sp
from scipy import stats, signal
import random
from sklearn import svm # needed for svm
from sklearn.metrics import confusion_matrix
import itertools
import pandas as pd

# Each accelerometer log file gets parsed and made into a SensorData object
class SensorData:

    # Constructors in Python look like this (strangely enough)
    # All arguments are numpy arrays except sensorType, which is a str
    def __init__(self, sensorType, currentTimeMs, sensorTimestampMs, x, y, z):
        self.sensorType = sensorType

        # On my mac, I could cast as straight-up int but on Windows, this failed
        # This is because on Windows, a long is 32 bit but on Unix, a long is 64bit
        # So, forcing to int64 to be safe. See: https://stackoverflow.com/q/38314118
        self.currentTimeMs = currentTimeMs.astype(np.int64)
```

AUTOMATIC FEATURE SELECTION

Because it's relatively easy to brainstorm and incorporate new features, it's tempting to just add everything (*i.e.*, increase the dimensionality)

However, adding more features increases the chance of overfitting, increases computational complexity, and requires more space/time to compute

SCIKIT FEATURE SELECTION API TO THE RESCUE

1.13. Feature selection

The classes in the `sklearn.feature_selection` module can be used for feature selection/dimensionality reduction on sample sets, either to improve estimators' accuracy scores or to boost their performance on very high-dimensional datasets.

Source: https://scikit-learn.org/stable/modules/feature_selection.html

AUTO-FEATURE SELECTION: MAIN APPROACHES

1. Analysis of variance. Simplest approach. Analyzes the variance of each feature across classes. Eliminates those with low variance.

2. Univariate statistics. Computes whether a feature has a statistically significant relationship with the classes (*i.e.*, is different from chance)

3. Model-based feature selection. Uses a supervised learning model (doesn't have to be the same one used for actual classification) to learn importance of features.

4. Iterative model-based feature selection. Extends #3 but runs a series of experiments either starting with a small number of features and growing or starting with the full feature set and removing.

Source: Chapter 2, Guido and Müller, *Introduction to Machine Learning with Python*, Oct 2016

AUTOMATIC FEATURE SELECTION

SOME SCIKIT LEARN VOCABULARY

Full Dataset (sans class labels)

In scikit-learn, the full dataset is referred to as **X**

	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag.mean	x.mean	y.mean	z.mean	mag_p.median	...	r
0	349	-0.003680	0.002231	-0.010517	0.001557	941.461182	487.939828	523.269341	611.925501	-0.002665	...	
1	329	0.006343	0.002568	0.005237	0.003226	941.442396	487.802432	523.452888	611.848024	0.009418	...	
2	266	0.007504	-0.005117	0.001735	0.014058	941.570441	487.887218	523.593985	611.845865	0.009391	...	
3	411	-0.001546	0.002486	0.001883	-0.005967	941.647837	487.922141	523.739659	611.822384	0.003814	...	
4	443	0.005105	-0.000706	0.005241	0.003945	941.695043	487.954853	523.819413	611.801354	0.013190	...	
5	224	1.781186	2.100282	0.081154	0.468409	978.420645	542.017857	508.205357	625.700893	-35.142362	...	
6	219	1.482731	1.722283	0.379335	0.206086	974.979163	535.164384	517.251142	621.908676	-17.023540	...	
7	234	1.341726	1.639692	0.260479	0.141812	974.949140	535.525641	521.311966	617.256410	-26.166495	...	
8	203	1.877071	2.107988	0.644584	0.149121	983.168663	545.261084	523.783251	618.669951	-29.787831	...	
9	254	1.146548	1.376192	0.228604	0.156053	976.387843	537.562992	521.389764	618.937008	-18.353688	...	
10	280	0.771997	1.154652	0.511057	-0.457802	971.279581	517.967857	567.910714	586.092857	-10.356645	...	
11	306	0.563156	0.672193	0.599782	-0.508517	980.382968	525.993464	580.692810	580.218954	-11.435248	...	
12	333	0.550743	0.633333	0.348090	-0.277344	994.310399	548.222222	584.675676	577.948949	-14.705581	...	
13	331	0.608548	0.629733	0.616082	-0.392998	989.807322	544.619335	573.797583	585.661631	-16.626730	...	
14	282	1.331867	1.640106	1.068490	-0.766823	1008.424162	567.460993	584.088652	580.439716	-23.186873	...	
15	357	0.197062	0.117094	0.133367	0.041783	949.635242	516.316527	505.490196	613.871148	-4.187680	...	
16	326	0.201396	0.126150	0.124161	0.055284	955.349994	510.987730	523.539877	612.647239	-3.785490	...	
17	376	0.139551	0.108374	0.110067	-0.008895	959.576622	513.441489	530.388298	611.348404	-3.820511	...	

...

Class Labels (aka ground truth labels)

In scikit-learn, the row labels are referred to as **y**

	gesture
0	At Rest
1	At Rest
2	At Rest
3	At Rest
4	At Rest
5	Backhand Tennis
6	Backhand Tennis
7	Backhand Tennis
8	Backhand Tennis
9	Backhand Tennis
10	Baseball Throw
11	Baseball Throw
12	Baseball Throw
13	Baseball Throw
14	Baseball Throw
15	Bunny Hops
16	Bunny Hops
17	Bunny Hops

...

AUTOMATIC FEATURE SELECTION

SOME SCIKIT LEARN VOCABULARY: FOUR TABLES

Training Features (sans class labels)

In scikit-learn, the training dataset is referred to as **X_train**

	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag.mean	x.mean	y.mean	z.mean	mag_p.median	...	y.max	z.max	mag_p.min	
44	169	0.504948	0.550432	0.081753	0.099924	992.399667	546.479290	538.485207	616.366864	-15.990339	...	792.0	706.0	-77.581025	-7
20	228	0.230991	0.198513	0.127168	0.026076	961.772456	514.964912	525.039474	613.337719	-15.258732	...	1008.0	744.0	-50.178799	-6
31	222	0.088741	0.088961	0.047581	-0.009783	949.266152	497.734234	529.851351	606.459459	-2.183821	...	713.0	715.0	-83.607727	-2
22	245	0.318951	0.279733	0.207738	-0.015133	972.575459	521.918367	532.897959	612.436735	-21.282488	...	1021.0	826.0	-72.391639	-8
10	280	0.238396	0.327967	0.180251	-0.125919	971.279581	517.967857	567.910714	586.092857	-10.342678	...	832.0	725.0	-71.217237	-9
17	376	0.043306	0.030411	0.034502	0.002339	959.576622	513.441489	530.388298	611.348404	-3.777288	...	648.0	891.0	-100.686142	-1
53	374	0.117810	0.185659	0.022076	-0.000018	949.936566	497.080214	539.002674	600.732620	-17.071119	...	713.0	690.0	-89.872787	-11
24	269	0.320089	0.310105	0.165845	0.021716	975.046147	525.970260	532.085502	615.988848	-25.573335	...	1017.0	785.0	-60.096884	-7
27	142	0.443843	0.539714	0.104087	0.025955	969.222639	533.985915	526.915493	604.056338	-10.714781	...	729.0	800.0	-48.356119	-6
23	255	0.232328	0.203235	0.159330	-0.021290	968.775038	517.862745	532.000000	612.819608	-19.214347	...	1020.0	799.0	-57.421396	-10
19	372	0.037929	0.040299	0.005110	0.004910	959.945478	512.612903	529.857527	612.080645	-5.584111	...	671.0	983.0	-106.830400	-1

...

Training Labels (aka ground truth labels)

In scikit-learn, the row labels for X_train are **y_train**

```
44          Midair Zorro 'Z'
20           Forehand Tennis
31       Midair Clockwise 'O'
22           Forehand Tennis
10            Baseball Throw
17                Bunny Hops
53        Underhand Bowling
24           Forehand Tennis
27                Midair 'S'
23           Forehand Tennis
19                Bunny Hops
...
```

Test Features (sans class labels)

In scikit-learn, the test dataset is referred to as **X_test**

	length	mag_p.mean	x_p.mean	y_p.mean	z_p.mean	mag.mean	x.mean	y.mean	z.mean	mag_p.median	...	y.max	z.max	mag_p.min	
50	309	0.192780	0.244329	0.105549	-0.012126	942.795756	501.702265	522.961165	599.391586	-13.299546	...	649.0	718.0	-72.286385	-108
2	266	0.003965	-0.003338	0.000780	0.008070	941.570441	487.887218	523.593985	611.845865	0.003803	...	527.0	622.0	-0.542440	-0
39	220	0.078192	0.078241	0.043304	-0.013675	951.967022	508.550000	516.809091	613.345455	-3.222963	...	623.0	758.0	-49.130342	-51
16	326	0.063253	0.043922	0.027338	0.026741	955.349994	510.987730	523.539877	612.647239	-3.784436	...	674.0	841.0	-92.309023	-13
34	196	0.231795	0.253251	0.075738	0.018883	959.160067	510.846939	527.362245	610.760204	-7.936969	...	796.0	744.0	-78.423893	-52
40	156	0.648326	0.631560	-0.006522	0.208427	981.757580	533.769231	518.166667	621.615385	-26.606337	...	843.0	767.0	-77.261739	-78
8	203	0.437270	0.445153	0.187909	0.032496	983.168663	545.261084	523.783251	618.669951	-30.349510	...	811.0	816.0	-70.112035	-80

...

Test Labels (aka ground truth labels)

In scikit-learn, the row labels for X_test are **y_test**

```
50          Underhand Bowling
2                     At Rest
39    Midair Counter-clockwise 'O'
16                Bunny Hops
34           Midair Clockwise 'O'
40                Midair Zorro 'Z'
8                  Backhand Tennis
...
```

AUTO-FEATURE SELECTION: MAIN APPROACHES

1. Analysis of variance. Simplest approach. Analyzes the variance of each feature across classes. Eliminates those with low variance.

2. Univariate statistics. Computes whether a feature has a statistically significant relationship with the classes (*i.e.*, is different from chance)

3. Model-based feature selection. Uses a supervised learning model (doesn't have to be the same one used for actual classification) to learn importance of features.

4. Iterative model-based feature selection. Extends #3 but runs a series of experiments either starting with a small number of features and growing or starting with the full feature set and removing.

Source: Chapter 2, Guido and Müller, *Introduction to Machine Learning with Python*, Oct 2016

Previous sklearn.featu ... | Next sklearn.featu ... | Up API Reference

scikit-learn v0.21.2
Other versions

Please cite us if you use the software.

sklearn.feature_selection.VarianceThreshold

sklearn.feature_selection.VarianceThreshold

class sklearn.feature_selection. **VarianceThreshold** (*threshold=0.0*) [source]

Feature selector that removes all low-variance features.

This feature selection algorithm looks only at the features (X), not the desired outputs (y), and can thus be used for unsupervised learning.

Read more in the User Guide.

Parameters:	**threshold : *float, optional*** Features with a training-set variance lower than this threshold will be removed. The default is to keep all features with non-zero variance, i.e. remove the features that have the same value in all samples.
Attributes:	**variances_ : *array, shape (n_features,)*** Variances of individual features.

Examples

The following dataset has integer features, two of which are the same in every sample. These are removed with the default setting for threshold:

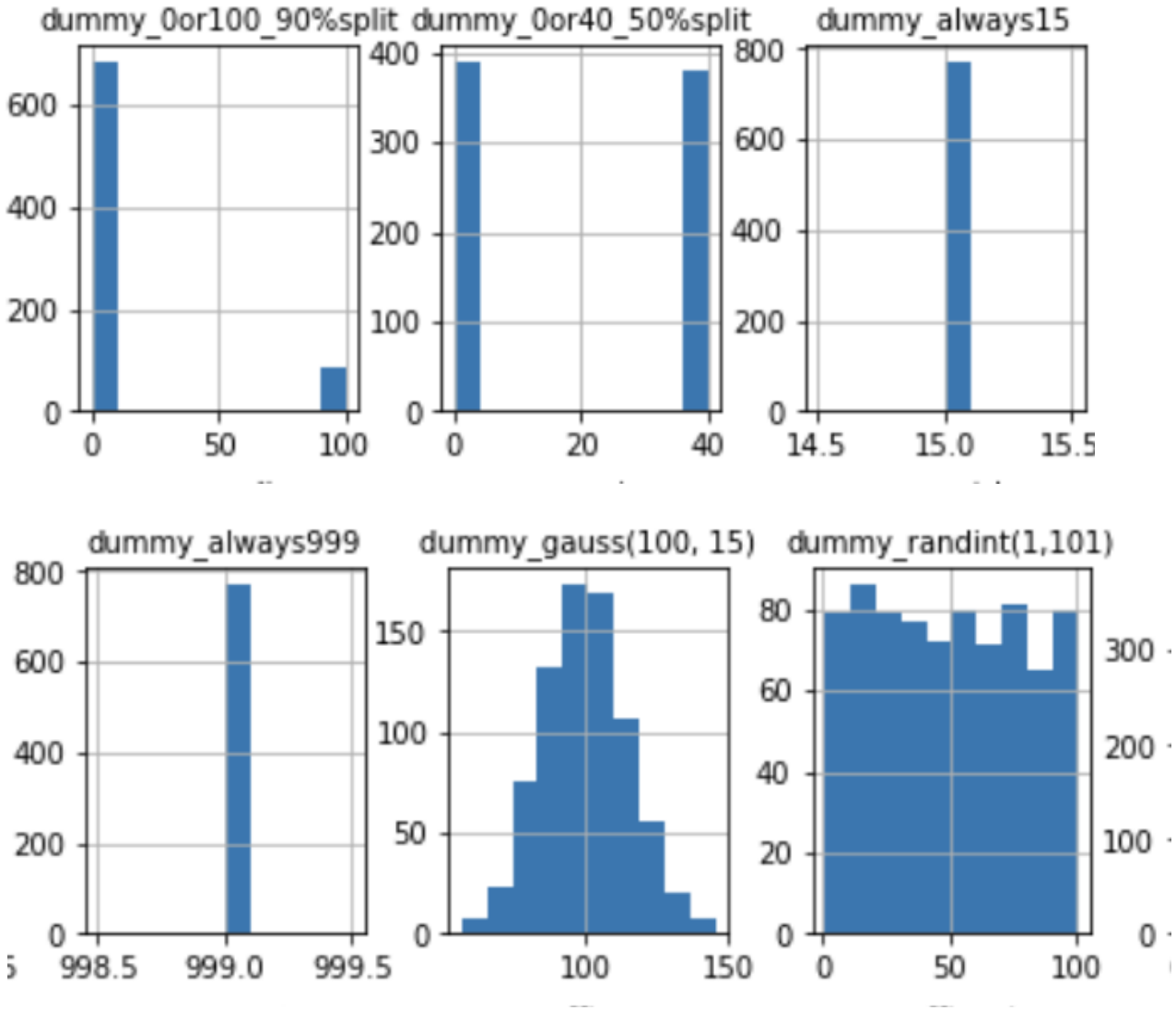
```
>>> X = [[0, 2, 0, 3], [0, 1, 4, 3], [0, 1, 1, 3]]
>>> selector = VarianceThreshold()
>>> selector.fit_transform(X)
array([[2, 0],
       [1, 4],
       [1, 1]])
```

Methods

FEATURE SELECTION

EXAMPLE IS WITH DUMMY FEATURES

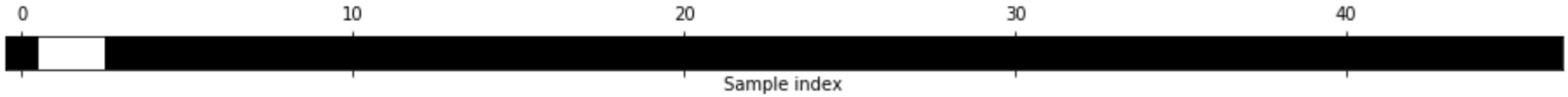
```
# dummy vars to test feature selection
if include_dummy_data:
    features.append(15)
    feature_names.append("dummy_always15")

    features.append(999)
    feature_names.append("dummy_always999")

    val0or40 = 0
    if random.random() > 0.5:
        val0or40 = 40
    features.append(val0or40)
    feature_names.append("dummy_0or40_50%split")

    val0or100 = 0
    if random.random() > 0.9:
        val0or100 = 100
    features.append(val0or100)
    feature_names.append("dummy_0or100_90%split")

    features.append(random.randint(1,101))
    feature_names.append("dummy_randint(1,101)")

    features.append(random.gauss(100, 15))
    feature_names.append("dummy_gauss(100, 15)")
```



```
df = pd.DataFrame(list_of_feature_vectors, columns = feature_names)
trial_indices = df.pop("trial_num") # pop off the trial numbers
X = df # set X to the dataframe
y = df.pop('gesture') # pop off the class labels
gesturer = df.pop('gesturer') # pop off the gesturer

# create a train/test set with 80% training, 20% test with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y)

# feature selection using VarianceThreshold
select = VarianceThreshold()
select.fit(X_train, y_train)

# transform training set
X_train_selected = select.transform(X_train)

# transform test data using the selector
X_test_selected = select.transform(X_test)

clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
print("Score with all features: {:.3f}".format(clf.score(X_test, y_test)))

clf.fit(X_train_selected, y_train)
print("Score with only selected features: {:.3f}".format(
    clf.score(X_test_selected, y_test)))
```

```
df = pd.DataFrame(list_of_feature_vectors, columns = feature_names)
trial_indices = df.pop("trial_num") # pop off the trial numbers
X = df # set X to the dataframe
y = df.pop('gesture') # pop off the class labels
gesturer = df.pop('gesturer') # pop off the gesturer

# create a train/test set with 80% training, 20% test with stratification
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y)

# feature selection using VarianceThreshold
select = VarianceThreshold()
select.fit(X_train, y_train)

# transform training set
X_train_selected = select.transform(X_train)

# transform test data using the selector
X_test_selected = select.transform(X_test)

clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
print("Score with all features: {:.3f}".format(clf.score(X_test, y_test)))

clf.fit(X_train_selected, y_train)
print("Score with only selected features: {:.3f}".format(
    clf.score(X_test_selected, y_test)))
```

```
X_train.shape: (44, 47)
X_train_selected.shape: (44, 45)
[ True False False  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True]
47 Index(['length', 'dummy_always15', 'dummy_always999', 'dummy_0or40_50%split',
       'dummy_0or100_90%split', 'dummy_randint(1,101)', 'dummy_gauss(100, 15)',
       'mag_p.mean', 'x_p.mean', 'y_p.mean', 'z_p.mean', 'mag.mean', 'x.mean',
       'y.mean', 'z.mean', 'mag_p.median', 'x_p.median', 'y_p.median',
       'z_p.median', 'mag.median', 'x.median', 'y.median', 'z.median',
       'mag_p.var', 'x_p.var', 'y_p.var', 'z_p.var', 'mag.var', 'x.var',
       'y.var', 'z.var', 'mag_p.max', 'x_p.max', 'y_p.max', 'z_p.max',
       'mag.max', 'x.max', 'y.max', 'z.max', 'mag_p.min', 'x_p.min', 'y_p.min',
       'z_p.min', 'mag.min', 'x.min', 'y.min', 'z.min'],
      dtype='object')
45 Index(['length', 'dummy_0or40_50%split', 'dummy_0or100_90%split',
       'dummy_randint(1,101)', 'dummy_gauss(100, 15)', 'mag_p.mean',
       'x_p.mean', 'y_p.mean', 'z_p.mean', 'mag.mean', 'x.mean', 'y.mean',
       'z.mean', 'mag_p.median', 'x_p.median', 'y_p.median', 'z_p.median',
       'mag.median', 'x.median', 'y.median', 'z.median', 'mag_p.var',
       'x_p.var', 'y_p.var', 'z_p.var', 'mag.var', 'x.var', 'y.var', 'z.var',
       'mag_p.max', 'x_p.max', 'y_p.max', 'z_p.max', 'mag.max', 'x.max',
       'y.max', 'z.max', 'mag_p.min', 'x_p.min', 'y_p.min', 'z_p.min',
       'mag.min', 'x.min', 'y.min', 'z.min'],
      dtype='object')
eliminated vars Index(['dummy_always15', 'dummy_always999'], dtype='object')
Score with all features: 0.909
Score with only selected features: 0.909
```



AUTO-FEATURE SELECTION: MAIN APPROACHES

1. Analysis of variance. Simplest approach. Analyzes the variance of each feature across classes. Eliminates those with low variance.

2. Univariate statistics. Computes whether a feature has a statistically significant relationship with the classes (*i.e.*, is different from chance)

3. Model-based feature selection. Uses a supervised learning model (doesn't have to be the same one used for actual classification) to learn importance of features.

4. Iterative model-based feature selection. Extends #3 but runs a series of experiments either starting with a small number of features and growing or starting with the full feature set and removing.

Source: Chapter 2, Guido and Müller, *Introduction to Machine Learning with Python*, Oct 2016

sklearn.feature_selection.SelectPercentile

class sklearn.feature_selection. **SelectPercentile** (*score_func=<function f_classif>*, *percentile=10*) [source]

Select features according to a percentile of the highest scores.

Read more in the User Guide.

Parameters:

score_func : *callable*

Function taking two arrays X and y, and returning a pair of arrays (scores, pvalues) or a single array with scores. Default is f_classif (see below “See also”). The default function only works with classification tasks.

percentile : *int, optional, default=10*

Percent of features to keep.

Attributes:

scores_ : *array-like, shape=(n_features,)*

Scores of features.

pvalues_ : *array-like, shape=(n_features,)*

p-values of feature scores, None if `score_func` returned only scores.

See also:

```
df = pd.DataFrame(list_of_feature_vectors, columns = feature_names)
trial_indices = df.pop("trial_num") # pop off the trial numbers
X = df # set X to the dataframe
y = df.pop('gesture') # pop off the class labels
gesturer = df.pop('gesturer') # pop off the gesturer

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, stratify=y)

# Scale data first
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# use f_classif (the default) and SelectPercentile to select some % of features
select = SelectPercentile(percentile=90)
select.fit(X_train, y_train)

# transform training set
X_train_selected = select.transform(X_train)

# transform test data
X_test_selected = select.transform(X_test)

clf = svm.SVC(kernel='linear')
clf.fit(X_train, y_train)
print("Score with all features: {:.3f}".format(clf.score(X_test, y_test)))

clf.fit(X_train_selected, y_train)
print("Score with only selected features: {:.3f}".format(
    clf.score(X_test_selected, y_test)))
```

```
X_train.shape: (44, 47)
X_train_selected.shape: (44, 42)
[ True False False  True False False  True  True  True  True  True  True
 False  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True  True
  True  True  True  True  True  True  True  True  True  True  True]
47 Index(['length', 'dummy_always15', 'dummy_always999', 'dummy_0or40_50%split',
       'dummy_0or100_90%split', 'dummy_randint(1,101)', 'dummy_gauss(100, 15)',
       'mag_p.mean', 'x_p.mean', 'y_p.mean', 'z_p.mean', 'mag.mean', 'x.mean',
       'y.mean', 'z.mean', 'mag_p.median', 'x_p.median', 'y_p.median',
       'z_p.median', 'mag.median', 'x.median', 'y.median', 'z.median',
       'mag_p.var', 'x_p.var', 'y_p.var', 'z_p.var', 'mag.var', 'x.var',
       'y.var', 'z.var', 'mag_p.max', 'x_p.max', 'y_p.max', 'z_p.max',
       'mag.max', 'x.max', 'y.max', 'z.max', 'mag_p.min', 'x_p.min', 'y_p.min',
       'z_p.min', 'mag.min', 'x.min', 'y.min', 'z.min'],
      dtype='object')
42 Index(['length', 'dummy_0or40_50%split', 'dummy_gauss(100, 15)', 'mag_p.mean',
       'x_p.mean', 'y_p.mean', 'z_p.mean', 'mag.mean', 'y.mean', 'z.mean',
       'mag_p.median', 'x_p.median', 'y_p.median', 'z_p.median', 'mag.median',
       'x.median', 'y.median', 'z.median', 'mag_p.var', 'x_p.var', 'y_p.var',
       'z_p.var', 'mag.var', 'x.var', 'y.var', 'z.var', 'mag_p.max', 'x_p.max',
       'y_p.max', 'z_p.max', 'mag.max', 'x.max', 'y.max', 'z.max', 'mag_p.min',
       'x_p.min', 'y_p.min', 'z_p.min', 'mag.min', 'x.min', 'y.min', 'z.min'],
      dtype='object')
eliminated vars Index(['dummy_always15', 'dummy_always999', 'dummy_0or100_90%split',
       'dummy_randint(1,101)', 'x.mean'],
      dtype='object')
Score with all features: 0.909
Score with only selected features: 0.909
```

AUTO-FEATURE SELECTION: MAIN APPROACHES

1. Analysis of variance. Simplest approach. Analyzes the variance of each feature across classes. Eliminates those with low variance.

2. Univariate statistics. Computes whether a feature has a statistically significant relationship with the classes (*i.e.*, is different from chance)

3. Model-based feature selection. Uses a supervised learning model (doesn't have to be the same one used for actual classification) to learn importance of features.

4. Iterative model-based feature selection. Extends #3 but runs a series of experiments either starting with a small number of features and growing or starting with the full feature set and removing.

Source: Chapter 2, Guido and Müller, *Introduction to Machine Learning with Python*, Oct 2016

Previous sklearn.featu ... | Next sklearn.featu ... | Up API Reference

scikit-learn v0.21.2
Other versions

Please cite us if you use the software.

sklearn.feature_selection.RFE
Examples using sklearn.feature_selection.RFE

sklearn.feature_selection.RFE

class sklearn.feature_selection. **RFE** (*estimator*, *n_features_to_select=None*, *step=1*, *verbose=0*) [source]

Feature ranking with recursive feature elimination.

Given an external estimator that assigns weights to features (e.g., the coefficients of a linear model), the goal of recursive feature elimination (RFE) is to select features by recursively considering smaller and smaller sets of features. First, the estimator is trained on the initial set of features and the importance of each feature is obtained either through a `coef_` attribute or through a `feature_importances_` attribute. Then, the least important features are pruned from current set of features. That procedure is recursively repeated on the pruned set until the desired number of features to select is eventually reached.

Read more in the User Guide.

Parameters:

estimator : *object*

A supervised learning estimator with a `fit` method that provides information about feature importance either through a `coef_` attribute or through a `feature_importances_` attribute.

n_features_to_select : *int or None (default=None)*

The number of features to select. If `None`, half of the features are selected.

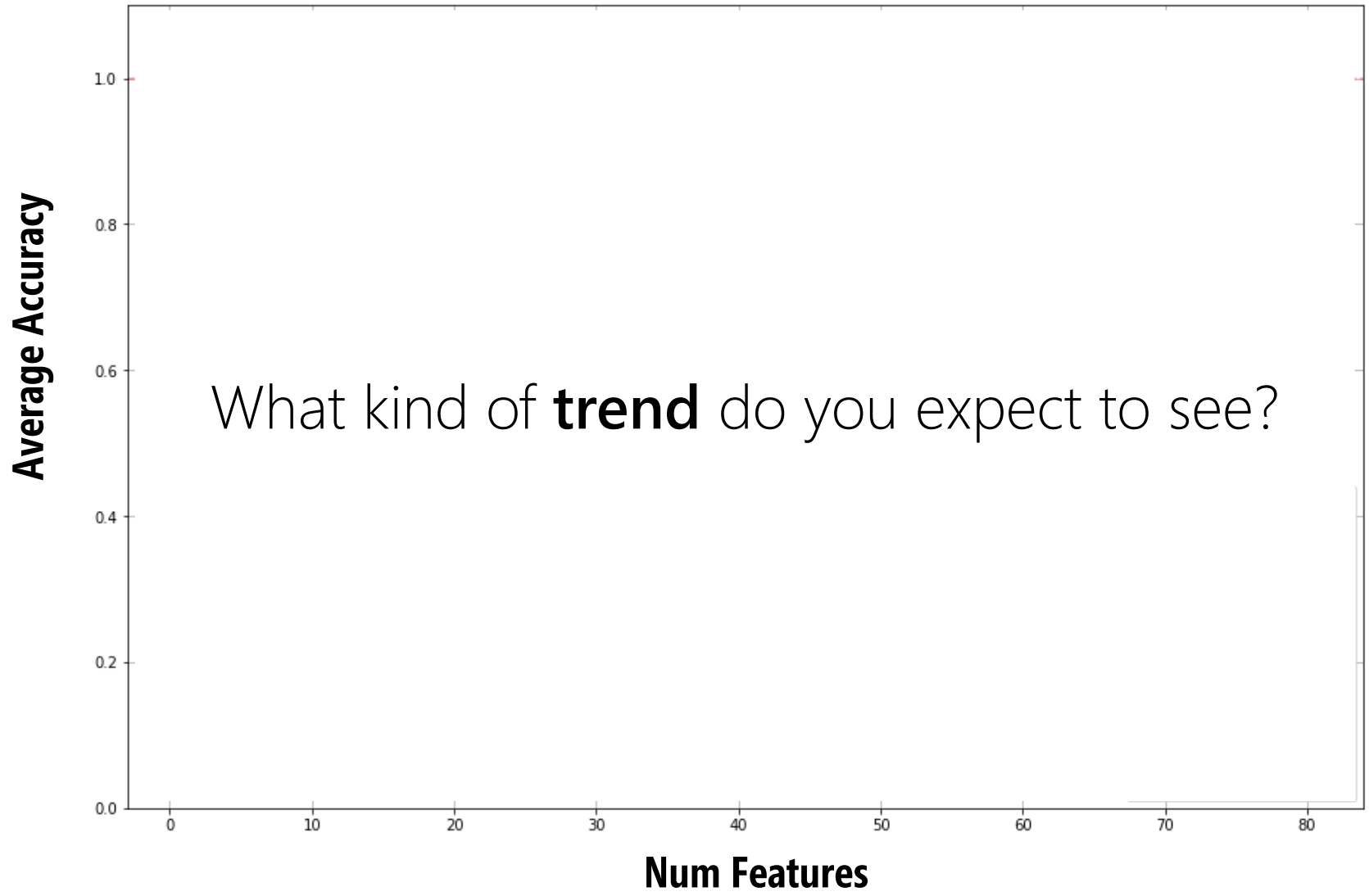
step : *int or float, optional (default=1)*

If greater than or equal to 1, then `step` corresponds to the (integer) number of features to re-

RFE EXPERIMENTS

ACCURACY AS A FUNCTION OF FEATURES

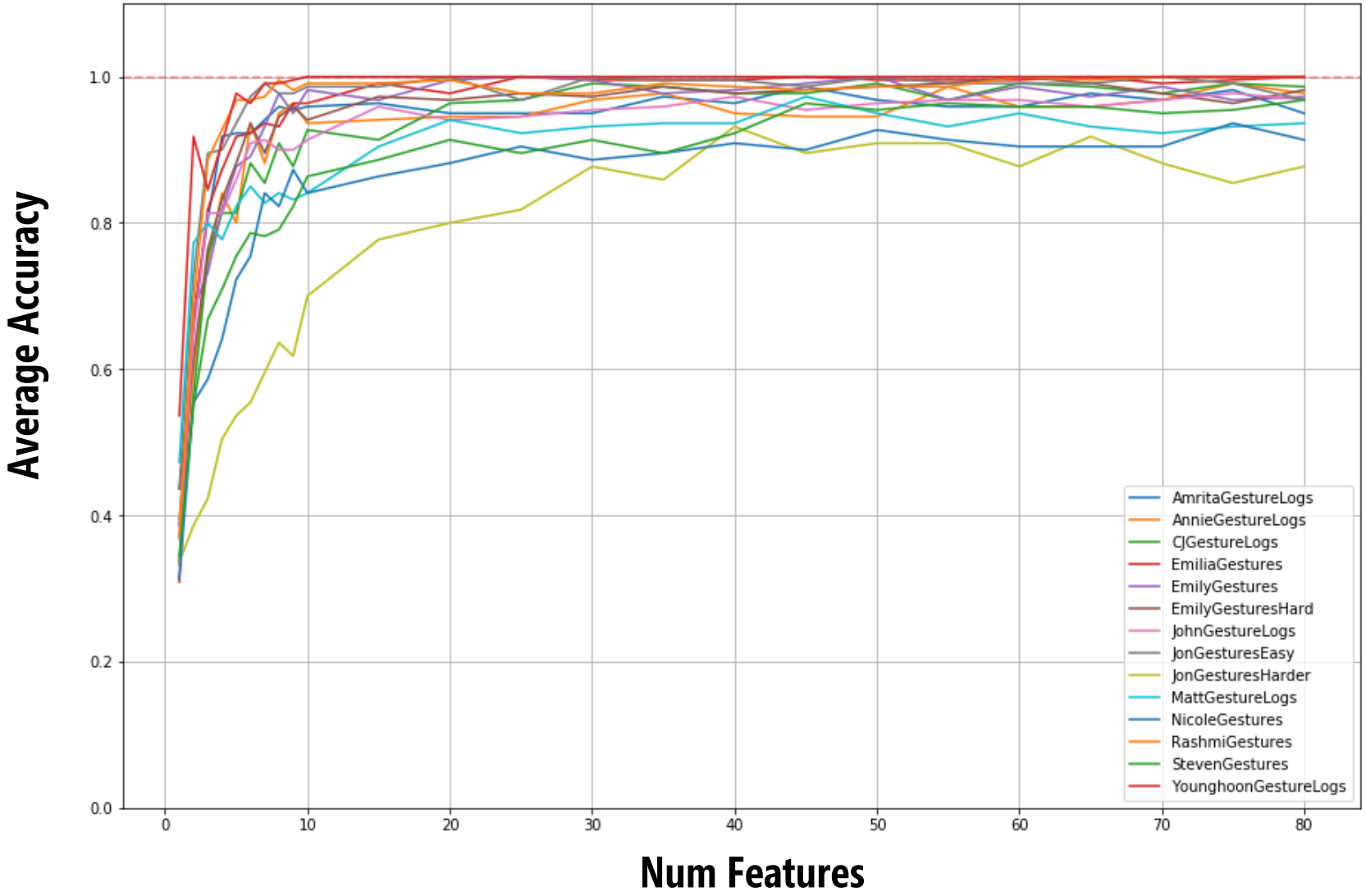
SVM.SVC(kernel='linear', C=0.1). RFE's min_features_to_select ranged from 1 to max; each tried 20 times



RFE EXPERIMENTS

ACCURACY AS A FUNCTION OF FEATURES

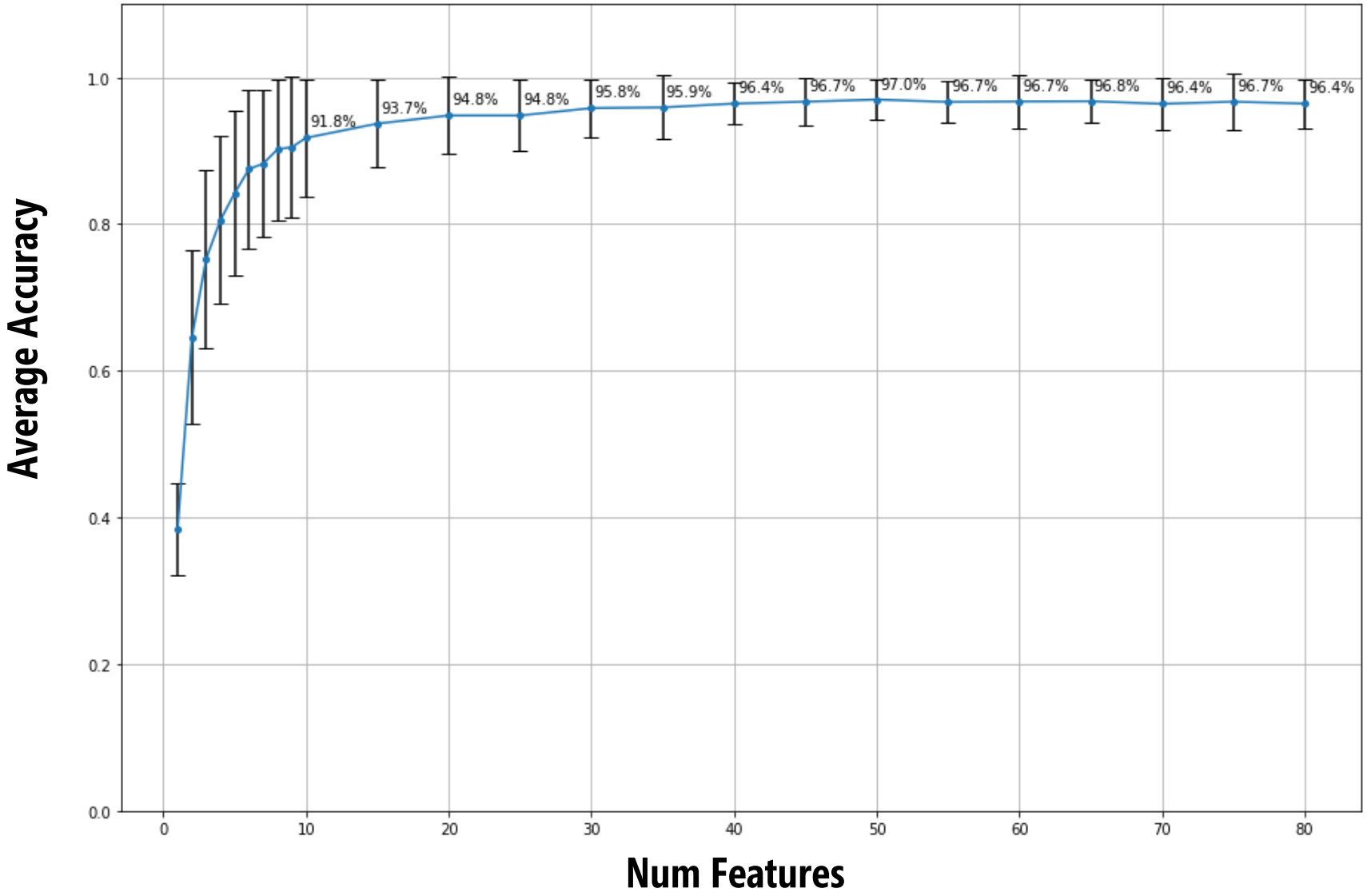
SVM.SVC(kernel='linear', C=0.1). RFE's min_features_to_select ranged from 1 to max; each tried 20 times



RFE EXPERIMENTS

ACCURACY AS A FUNCTION OF FEATURES

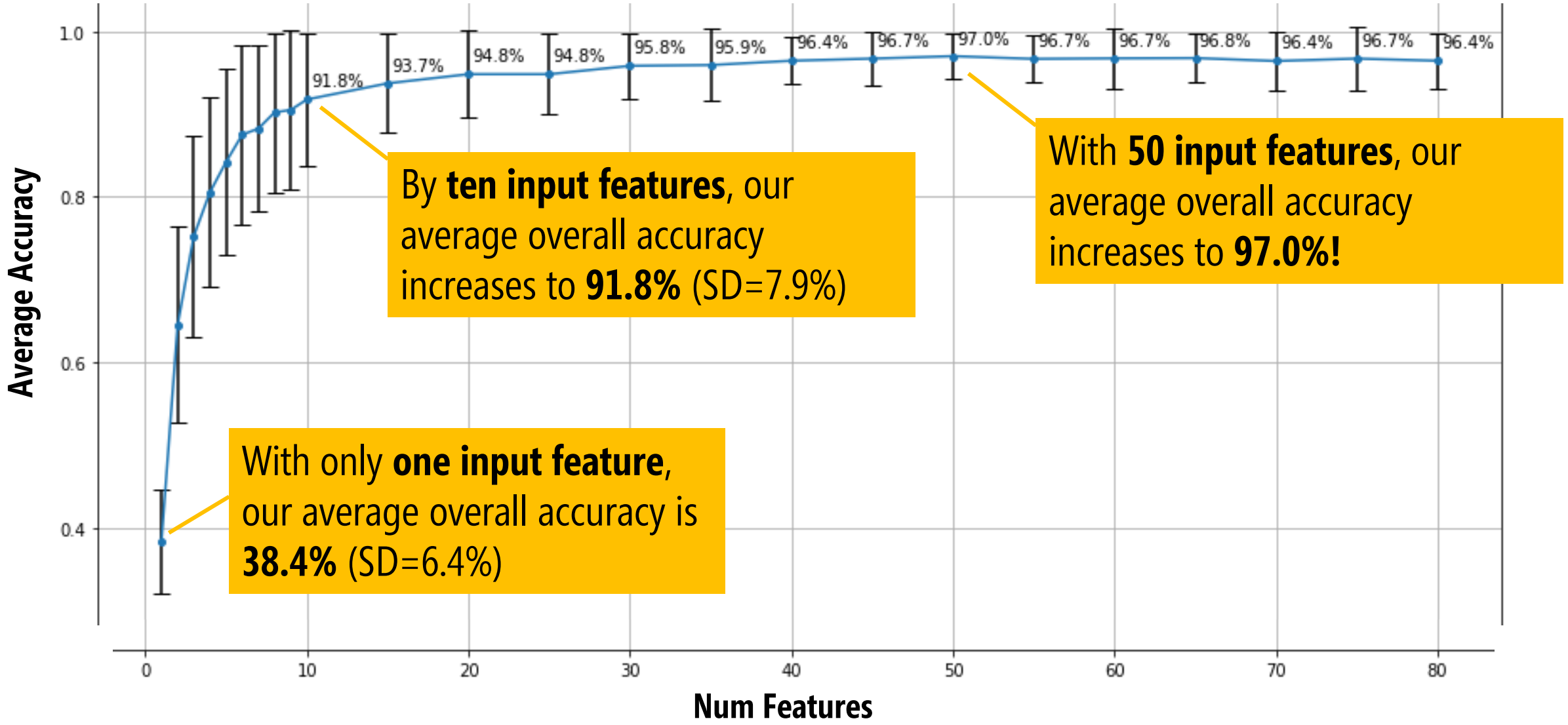
SVM.SVC(kernel='linear', C=0.1). RFE's min_features_to_select ranged from 1 to max; each tried 20 times



RFE EXPERIMENTS

ACCURACY AS A FUNCTION OF FEATURES

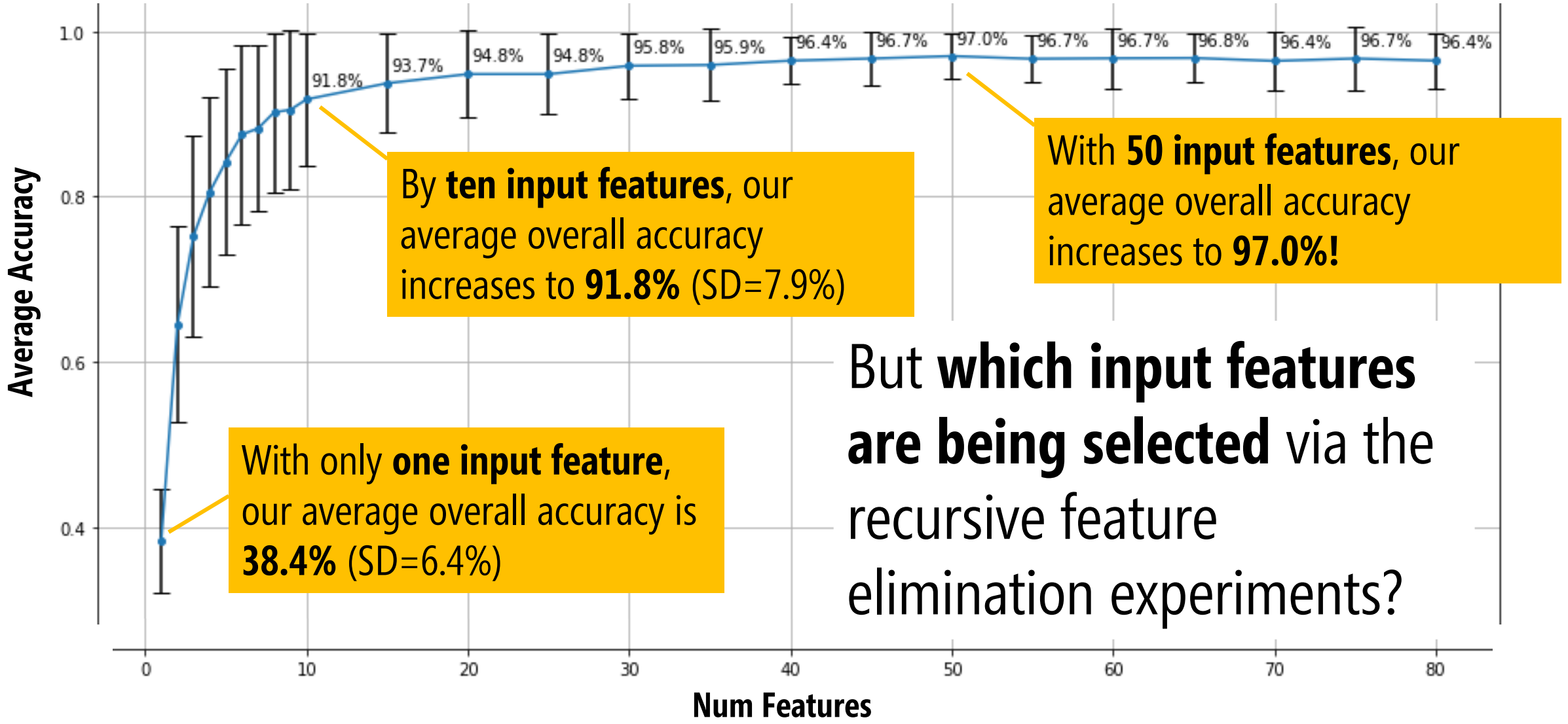
SVM.SVC(kernel='linear', C=0.1). RFE's min_features_to_select ranged from 1 to max; each tried 20 times



RFE EXPERIMENTS

ACCURACY AS A FUNCTION OF FEATURES

SVM.SVC(kernel='linear', C=0.1). RFE's min_features_to_select ranged from 1 to max; each tried 20 times



By **ten input features**, our average overall accuracy increases to **91.8%** (SD=7.9%)

With **50 input features**, our average overall accuracy increases to **97.0%!**

With only **one input feature**, our average overall accuracy is **38.4%** (SD=6.4%)

But **which input features are being selected** via the recursive feature elimination experiments?

RFE EXPERIMENTS

RFE PROVIDES LOTS OF METADATA

Attributes:

n_features_ : *int*

The number of selected features.

support_ : *array of shape [n_features]*

The mask of selected features.

ranking_ : *array of shape [n_features]*

The feature ranking, such that `ranking_[i]` corresponds to the ranking position of the i-th feature. Selected (i.e., estimated best) features are assigned rank 1.

estimator_ : *object*

The external estimator fit on the reduced dataset.

In the experiment, I kept track of **which features were used** and **which were discarded** at each feature count threshold across all gesture sets.

```
20 FEATURES: OVERALL ACCURACY 90.65% (SD=9.61%)
        FEATURES USED:
        z.mean : (181/280) 64.6%
        y.mean : (173/280) 61.8%
        y.median : (162/280) 57.9%
        y_p.min : (158/280) 56.4%
        y_p.std : (155/280) 55.4%
        mag_p.peak_count : (151/280) 53.9%
        x.mean : (144/280) 51.4%
        x.max : (136/280) 48.6%
        y_p.peak_count : (133/280) 47.5%
        mag.mean : (132/280) 47.1%
        y.max : (126/280) 45.0%
        z.median : (122/280) 43.6%
        x.median : (120/280) 42.9%
        y_p.max : (113/280) 40.4%
        y_p.median : (110/280) 39.3%
        z.max : (110/280) 39.3%
        z_p.std : (108/280) 38.6%
        x_p.peak_avg : (108/280) 38.6%
        y.min : (107/280) 38.2%
        mag_p.peak_avg : (105/280) 37.5%
        mag.median : (103/280) 36.8%
        x_p.fft_entropy : (103/280) 36.8%
        z_p.fft_entropy : (102/280) 36.4%
        mag.min : (96/280) 34.3%
        x_p.max : (96/280) 34.3%
        x.std : (94/280) 33.6%
        y_p.var : (93/280) 33.2%
        z.min : (92/280) 32.9%
        y_p.fft_entropy : (92/280) 32.9%
        z_p.peak_count : (89/280) 31.8%
        z_p.max : (85/280) 30.4%
        z_p.median : (84/280) 30.0%
        x.min : (82/280) 29.3%
        y.std : (77/280) 27.5%
        x_p.std : (76/280) 27.1%
        z_p.min : (75/280) 26.8%
        length : (75/280) 26.8%
        x_p.median : (70/280) 25.0%
        mag_p.min : (70/280) 25.0%
        x_p.mean : (68/280) 24.3%
        y_p.peak_avg : (68/280) 24.3%
        y_p.fft_energy : (62/280) 22.1%
        z_p.peak_avg : (61/280) 21.8%
        x_p.min : (61/280) 21.8%
        mag_p.median : (57/280) 20.4%
        x.var : (51/280) 18.2%
        mag.max : (49/280) 17.5%
        z_p.var : (48/280) 17.1%
        mag_p.max_freq : (47/280) 16.8%
        x_p.peak_count : (44/280) 15.7%
        mag_p.mean : (43/280) 15.4%
        mag.var : (42/280) 15.0%
        mag_p.zero_crossings : (41/280) 14.6%
        mag.std : (40/280) 14.3%
        z.std : (40/280) 14.3%
        z_p.mean : (38/280) 13.6%
        y_p.max_freq : (32/280) 11.4%
        x_p.zero_crossings : (31/280) 11.1%
        mag_p.var : (28/280) 10.0%
        mag_p.max : (28/280) 10.0%
        mag_p.top2_freq : (27/280) 9.6%
        x_p.max_freq : (27/280) 9.6%
        mag_p.top2_freq_coef : (26/280) 9.3%
        mag_p.top1_freq_coef : (26/280) 9.3%
        x_p.var : (25/280) 8.9%
        mag_p.top1_freq : (23/280) 8.2%
        mag_p.top3_freq_coef : (23/280) 8.2%
        y.var : (22/280) 7.9%
        y_p.mean : (22/280) 7.9%
        z_p.max_freq : (20/280) 7.1%
        mag_p.std : (18/280) 6.4%
        z.var : (17/280) 6.1%
        mag_p.fft_entropy : (13/280) 4.6%
        mag_p.fft_energy : (12/280) 4.3%
        z_p.fft_energy : (6/280) 2.1%
        x_p.fft_energy : (4/280) 1.4%
        z_p.zero_crossings : (2/280) 0.7%
```

10 FEATURES: OVERALL ACCURACY 91.75% (SD=7.92%)

FEATURES USED RATES:

00 y.mean : (173/280) 61.8%
01 y_p.min : (104/280) 37.1%
02 z.mean : (99/280) 35.4%
03 y_p.median : (95/280) 33.9%
04 y_p.peak_count : (93/280) 33.2%
05 z_p.peak_count : (88/280) 31.4%
06 x.mean : (87/280) 31.1%
07 y_p.max : (85/280) 30.4%
08 x.max : (85/280) 30.4%
09 z_p.peak_avg : (84/280) 30.0%
10 y.median : (82/280) 29.3%
11 mag.mean : (79/280) 28.2%
12 z_p.median : (77/280) 27.5%
13 z.median : (72/280) 25.7%
14 z_p.min : (72/280) 25.7%
15 x_p.fft_entropy : (66/280) 23.6%
16 x_p.mean : (65/280) 23.2%
17 x.min : (63/280) 22.5%
18 x_p.peak_avg : (61/280) 21.8%
19 x.median : (58/280) 20.7%
20 x_p.max : (56/280) 20.0%
21 y_p.peak_avg : (48/280) 17.1%
22 mag.median : (48/280) 17.1%
23 mag_p.peak_count : (47/280) 16.8%
24 y_p.mean : (46/280) 16.4%
25 y_p.fft_entropy : (44/280) 15.7%
26 y.max : (44/280) 15.7%
27 y_p.std : (38/280) 13.6%
28 z_p.std : (38/280) 13.6%
29 z_p.max : (36/280) 12.9%
30 y.min : (33/280) 11.8%
31 mag_p.median : (32/280) 11.4%
32 mag_p.peak_avg : (32/280) 11.4%
33 length : (31/280) 11.1%
34 z_p.max_freq : (31/280) 11.1%
35 z_p.var : (31/280) 11.1%
36 y_p.var : (30/280) 10.7%
37 mag.min : (29/280) 10.4%
38 x_p.peak_count : (28/280) 10.0%
39 x_p.min : (26/280) 9.3%
40 x.std : (24/280) 8.6%
41 x_p.median : (23/280) 8.2%
42 z_p.fft_entropy : (23/280) 8.2%
43 x.var : (23/280) 8.2%
44 x_p.std : (21/280) 7.5%
45 mag_p.top1_freq_coef : (20/280) 7.1%

FEATURES DISCARDED RATES

00 mag_p.std : (280/280) 100.0%
01 z.std : (280/280) 100.0%
02 z.var : (280/280) 100.0%
03 mag_p.fft_entropy : (280/280) 100.0%
04 x_p.fft_energy : (280/280) 100.0%
05 mag_p.var : (279/280) 99.6%
06 mag_p.top3_freq : (279/280) 99.6%
07 z_p.fft_energy : (279/280) 99.6%
08 z_p.zero_crossings : (278/280) 99.3%
09 mag_p.max_freq : (278/280) 99.3%
10 mag_p.top1_freq : (278/280) 99.3%
11 mag_p.fft_energy : (278/280) 99.3%
12 y.std : (277/280) 98.9%
13 x_p.max_freq : (277/280) 98.9%
14 y_p.max_freq : (277/280) 98.9%
15 y.var : (276/280) 98.6%
16 x_p.zero_crossings : (276/280) 98.6%
17 mag_p.max : (275/280) 98.2%
18 mag.max : (274/280) 97.9%
19 y_p.fft_energy : (274/280) 97.9%
20 mag_p.zero_crossings : (272/280) 97.1%
21 z_p.mean : (270/280) 96.4%
22 x_p.var : (269/280) 96.1%
23 mag_p.top2_freq_coef : (269/280) 96.1%
24 mag_p.min : (266/280) 95.0%
25 y_p.zero_crossings : (266/280) 95.0%
26 mag_p.top2_freq : (265/280) 94.6%
27 z.min : (264/280) 94.3%
28 mag.std : (263/280) 93.9%
29 mag.var : (263/280) 93.9%
30 z.max : (263/280) 93.9%
31 mag_p.top3_freq_coef : (263/280) 93.9%
32 mag_p.mean : (262/280) 93.6%
33 mag_p.top1_freq_coef : (260/280) 92.9%
34 x_p.std : (259/280) 92.5%
35 x.var : (257/280) 91.8%
36 z_p.fft_entropy : (257/280) 91.8%
37 x_p.median : (257/280) 91.8%
38 x.std : (256/280) 91.4%
39 x_p.min : (254/280) 90.7%
40 x_p.peak_count : (252/280) 90.0%
41 mag.min : (251/280) 89.6%
42 y_p.var : (250/280) 89.3%
43 length : (249/280) 88.9%
44 z_p.var : (249/280) 88.9%
45 z_p.max_freq : (249/280) 88.9%
46 mag_p.median : (248/280) 88.6%
47 mag_p.peak_avg : (248/280) 88.6%
48 y.min : (247/280) 88.2%
49 z_p.max : (244/280) 87.1%
50 z_p.std : (242/280) 86.4%
51 y_p.std : (242/280) 86.4%
52 y.max : (236/280) 84.3%
53 y_p.fft_entropy : (236/280) 84.3%
54 y_p.mean : (234/280) 83.6%
55 mag_p.peak_count : (233/280) 83.2%
56 mag.median : (232/280) 82.9%
57 y_p.peak_avg : (232/280) 82.9%
58 x_p.max : (224/280) 80.0%
59 x.median : (222/280) 79.3%
60 x_p.peak_avg : (219/280) 78.2%
61 x.min : (217/280) 77.5%
62 x_p.mean : (215/280) 76.8%
63 x_p.fft_entropy : (214/280) 76.4%
64 z_p.min : (208/280) 74.3%
65 z.median : (208/280) 74.3%
66 z_p.median : (203/280) 72.5%
67 mag.mean : (201/280) 71.8%
68 y.median : (198/280) 70.7%
69 z_p.peak_avg : (196/280) 70.0%
70 x.max : (195/280) 69.6%
71 y_p.max : (195/280) 69.6%
72 x.mean : (193/280) 68.9%
73 z_p.peak_count : (192/280) 68.6%
74 y_p.peak_count : (187/280) 66.8%
75 y_p.median : (185/280) 66.1%
76 z.mean : (181/280) 64.6%
77 y_p.min : (176/280) 62.9%
78 y.mean : (107/280) 38.2%

RFE EXPERIMENTS

WITH MAX INPUT FEATURES=1: 38.4% ACCURACY

For each gesture set, we ran 20 RFE experiments with the feature count set to 1

```
1 FEATURES: OVERALL ACCURACY 38.41% (SD
        FEATURES USED RATES:
        00 y.mean : (29/280) 10.4%
        01 y_p.max : (24/280) 8.6%
        02 z.median : (21/280) 7.5%
        03 x.max : (21/280) 7.5%
        04 z_p.peak_avg : (21/280) 7.5%
        05 y_p.peak_count : (17/280) 6.1%
        06 y_p.std : (13/280) 4.6%
        07 y_p.min : (13/280) 4.6%
        08 x.mean : (11/280) 3.9%
        09 x.min : (10/280) 3.6%
        10 mag_p.top3_freq_coef : (9/280) 3.2%
```

Of the **280 RFE executions** (20 per gesture set) with the feature count set to one, **10.4%** of those used **y_mean** as the sole input feature

```
                                  30) 3.2%
                                 280) 2.9%
                                 /280) 2.5%
        14 y.max : (7/280) 2.5%
        15 x_p.std : (6/280) 2.1%
        16 mag_p.peak_avg : (6/280) 2.1%
        17 z_p.peak_count : (5/280) 1.8%
        18 mag_p.mean : (5/280) 1.8%
        19 y_p.peak_avg : (5/280) 1.8%
        20 x.std : (3/280) 1.1%
        21 mag.median : (3/280) 1.1%
```

...

(long tail)

RFE EXPERIMENTS

WITH MAX INPUT FEATURES=10: 91.8% ACCURACY

For each gesture set, we ran 20 RFE experiments with the max number of features set to 10

```
10 FEATURES: OVERALL ACCURACY 91.75% (SD=7
        FEATURES USED RATES:
        00 y.mean : (173/280) 61.8%
        01 y_p.min : (104/280) 37.1%
        02 z.mean : (99/280) 35.4%
        03 y_p.median : (95/280) 33.9%
        04 y_p.peak_count : (93/280) 33.2%
        05 z_p.peak_count : (88/280) 31.4%
        06 x.mean : (87/280) 31.1%
        07 y_p.max : (85/280) 30.4%
        08 x.max : (85/280) 30.4%
        09 z_p.peak_avg : (84/280) 30.0%
        10 y.median : (82/280) 29.3%
        11 mag.mean : (79/280) 28.2%
        12 z_p.median : (77/280) 27.5%
        13 z.median : (72/280) 25.7%
        14 z_p.min : (72/280) 25.7%
        15 x_p.fft_entropy : (66/280) 23.6%
```

Of the **280 RFE executions** (20 per gesture set) with input feature limit set to ten, **61.8%** of those used **y_mean** as an input feature

~33% of those used **y_p and z_p peak counts**

RFE EXPERIMENTS

WITH MAX INPUT FEATURES=10: 91.8% ACCURACY

For each gesture set, we ran 20 RFE experiments with the max number of features set to 10

```
10 FEATURES: OVERALL ACCURACY 91.75% (SD=7.92%)
        FEATURES USED RATES:
        00 y.mean : (173/280) 61.8%
        01 y_p.min : (104/280) 37.1%
        02 z.mean : (99/280) 35.4%
        03 y_p.median : (95/280) 33.9%
        04 y_p.peak_count : (93/280) 33.2%
        05 z_p.peak_count : (88/280) 31.4%
        06 x.mean : (87/280) 31.1%
        07 y_p.max : (85/280) 30.4%
        08 x.max : (85/280) 30.4%
        09 z_p.peak_avg : (84/280) 30.0%
        10 y.median : (82/280) 29.3%
        11 mag.mean : (79/280) 28.2%
        12 z_p.median : (77/280) 27.5%
        13 z.median : (72/280) 25.7%
        14 z_p.min : (72/280) 25.7%
        15 x_p.fft_entropy : (66/280) 23.6%
```

```
FEATURES DISCARDED RATES
00 mag_p.std : (280/280) 100.0%
01 z.std : (280/280) 100.0%
02 z.var : (280/280) 100.0%
03 mag_p.fft_entropy : (280/280) 100.0%
04 x_p.fft_energy : (280/280) 100.0%
05 mag_p.var : (279/280) 99.6%
06 mag_p.top3_freq : (279/280) 99.6%
07 z_p.fft_energy : (279/280) 99.6%
08 z_p.zero_crossings : (278/280) 99.3%
09 mag_p.max_freq : (278/280) 99.3%
10 mag_p.top1_freq : (278/280) 99.3%
11 mag_p.fft_energy : (278/280) 99.3%
12 y.std : (277/280) 98.9%
13 x_p.max_freq : (277/280) 98.9%
14 y_p.max_freq : (277/280) 98.9%
15 y.var : (276/280) 98.6%
```

RFE EXPERIMENTS

WITH MAX INPUT FEATURES=10: 91.8% ACCURACY

For each gesture set, we ran 20 RFE experiments with the max number of features set to 10

```
10 FEATURES: OVERALL ACCURACY 91.75% (SD=7.92%)
        FEATURES USED RATES:
        00 y.mea
        01 y_p.m
        02 z.mea
        03 y_p.m
        04 y_p.p
        05 z_p.p
        06 x.mea
        07 y_p.max : (85/280) 30.4%
        08 x.max : (85/280) 30.4%
        09 z_p.peak_avg : (84/280) 30.0%
        10 y.median : (82/280) 29.3%
        11 mag.mean : (79/280) 28.2%
        12 z_p.median : (77/280) 27.5%
        13 z.median : (72/280) 25.7%
        14 z_p.min : (72/280) 25.7%
        15 x_p.fft_entropy : (66/280) 23.6%
```

Of the **280 RFE executions** (20 per gesture set) with input feature limit set to ten, **these five features** were always discarded

```
FEATURES DISCARDED RATES
00 mag_p.std : (280/280) 100.0%
01 z.std : (280/280) 100.0%
02 z.var : (280/280) 100.0%
03 mag_p.fft_entropy : (280/280) 100.0%
04 x_p.fft_energy : (280/280) 100.0%
05 mag_p.var : (279/280) 99.6%
06 mag_p.top3_freq : (279/280) 99.6%
07 z_p.fft_energy : (279/280) 99.6%
08 z_p.zero_crossings : (278/280) 99.3%
09 mag_p.max_freq : (278/280) 99.3%
10 mag_p.top1_freq : (278/280) 99.3%
11 mag_p.fft_energy : (278/280) 99.3%
12 y.std : (277/280) 98.9%
13 x_p.max_freq : (277/280) 98.9%
14 y_p.max_freq : (277/280) 98.9%
15 y.var : (276/280) 98.6%
```

RFE EXPERIMENTS

WITH MAX INPUT FEATURES=30: 95.8% ACCURACY

For each gesture set, we ran 20 RFE experiments with the max number of features set to 30

```
30 FEATURES: OVERALL ACCURACY 95.81% (SD=4.01%)
        FEATURES USED RATES:
        00 y.mean : (259/280) 92.5%
        01 y.median : (255/280) 91.1%
        02 x_p.fft_entropy : (206/280) 73.6%
        03 y_p.mean : (205/280) 73.2%
        04 y_p.min : (204/280) 72.9%
        05 z_p.median : (202/280) 72.1%
        06 y_p.peak_avg : (195/280) 69.6%
        07 y_p.median : (193/280) 68.9%
        08 y_p.max : (192/280) 68.6%
        09 x.median : (190/280) 67.9%
        10 z_p.peak_count : (188/280) 67.1%
        11 x.min : (187/280) 66.8%
        12 z.mean : (186/280) 66.4%
        13 y_p.peak_count : (183/280) 65.4%
        14 mag_p.peak_count : (175/280) 62.5%
        15 z_p.peak_avg : (174/280) 62.1%
```

```
FEATURES DISCARDED RATES
00 mag_p.std : (276/280) 98.6%
01 x_p.fft_energy : (276/280) 98.6%
02 mag_p.var : (268/280) 95.7%
03 mag_p.fft_energy : (265/280) 94.6%
04 y_p.fft_energy : (260/280) 92.9%
05 mag.std : (256/280) 91.4%
06 z.var : (256/280) 91.4%
07 mag_p.max : (255/280) 91.1%
08 y.std : (255/280) 91.1%
09 mag_p.top1_freq : (248/280) 88.6%
10 z.std : (247/280) 88.2%
11 x_p.var : (247/280) 88.2%
12 mag_p.zero_crossings : (247/280) 88.2%
13 y.var : (246/280) 87.9%
14 y_p.max_freq : (245/280) 87.5%
15 mag.var : (240/280) 85.7%
```

RFE EXPERIMENTS

WITH MAX INPUT FEATURES=60: 96.7% ACCURACY

For each gesture set, we ran 20 RFE experiments with the max number of features set to 60

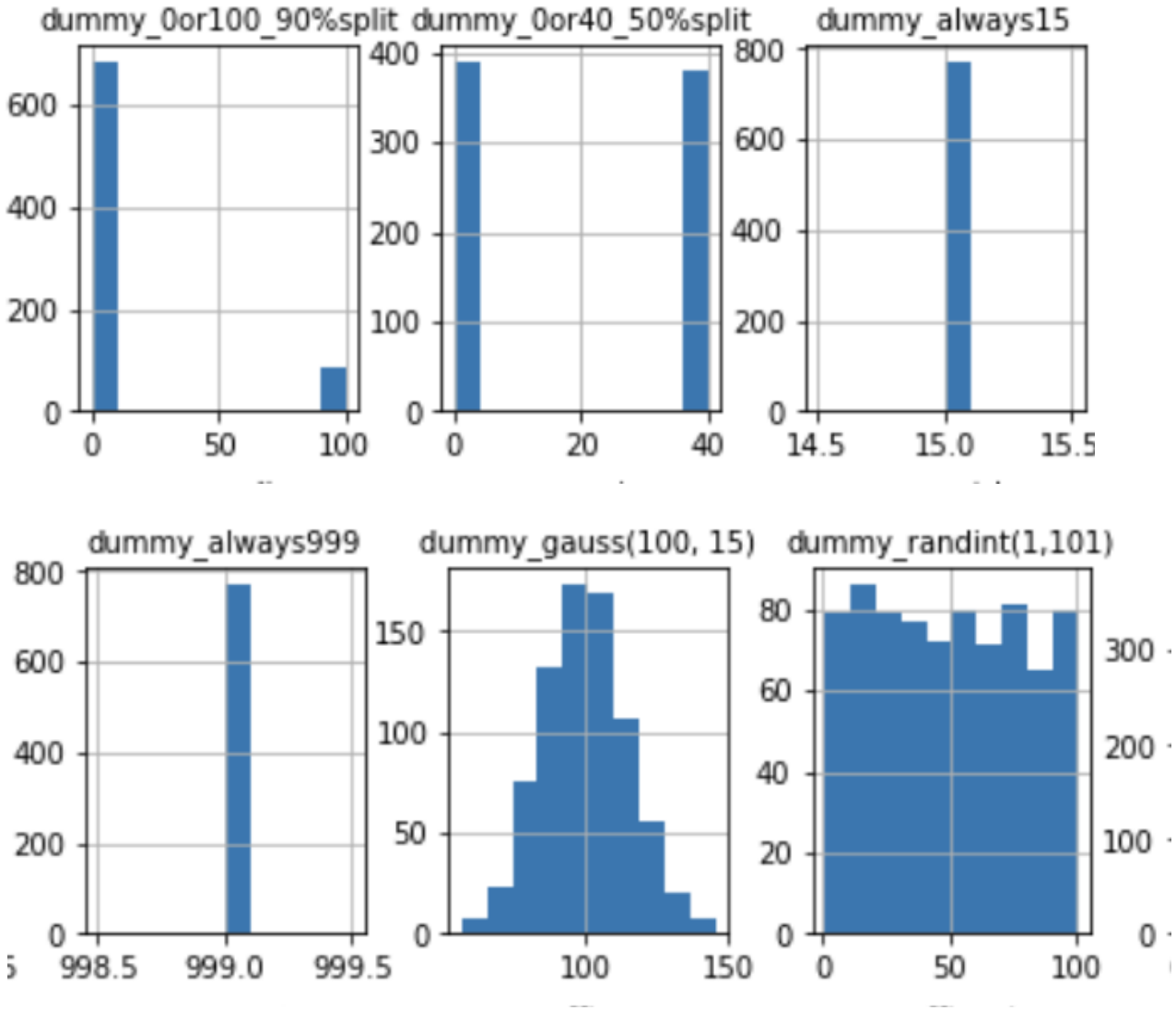
```
60 FEATURES: OVERALL ACCURACY 96.72% (SD=3.59%)
        FEATURES USED RATES:
        00 y.mean : (280/280) 100.0%
        01 y_p.median : (280/280) 100.0%
        02 y.median : (280/280) 100.0%
        03 y_p.std : (280/280) 100.0%
        04 y_p.min : (280/280) 100.0%
        05 z_p.median : (278/280) 99.3%
        06 y.min : (277/280) 98.9%
        07 z_p.peak_avg : (277/280) 98.9%
        08 mag_p.peak_count : (276/280) 98.6%
        09 y_p.peak_count : (272/280) 97.1%
        10 z_p.fft_entropy : (272/280) 97.1%
        11 length : (270/280) 96.4%
        12 x.min : (269/280) 96.1%
        13 x.median : (268/280) 95.7%
        14 y_p.max : (267/280) 95.4%
        15 z_p.peak_count : (264/280) 94.3%
```

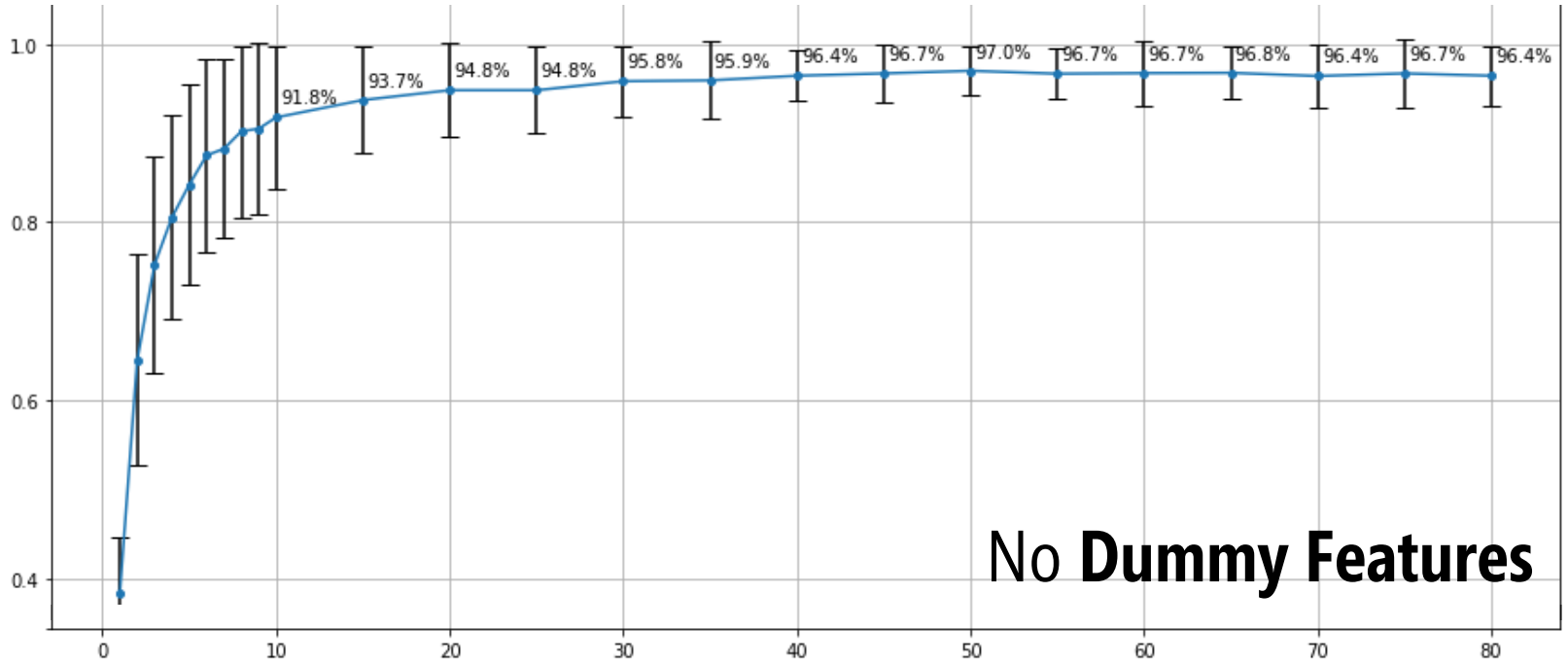
```
FEATURES DISCARDED RATES
00 mag.var : (230/280) 82.1%
01 mag.std : (213/280) 76.1%
02 mag_p.fft_energy : (212/280) 75.7%
03 z.var : (198/280) 70.7%
04 x_p.fft_energy : (191/280) 68.2%
05 mag_p.var : (178/280) 63.6%
06 mag_p.std : (172/280) 61.4%
07 y.var : (167/280) 59.6%
08 z_p.fft_energy : (159/280) 56.8%
09 x_p.var : (145/280) 51.8%
10 mag_p.zero_crossings : (144/280) 51.4%
11 z_p.var : (139/280) 49.6%
12 x.var : (132/280) 47.1%
13 y_p.max_freq : (127/280) 45.4%
14 z_p.zero_crossings : (126/280) 45.0%
15 y_p.fft_energy : (122/280) 43.6%
```

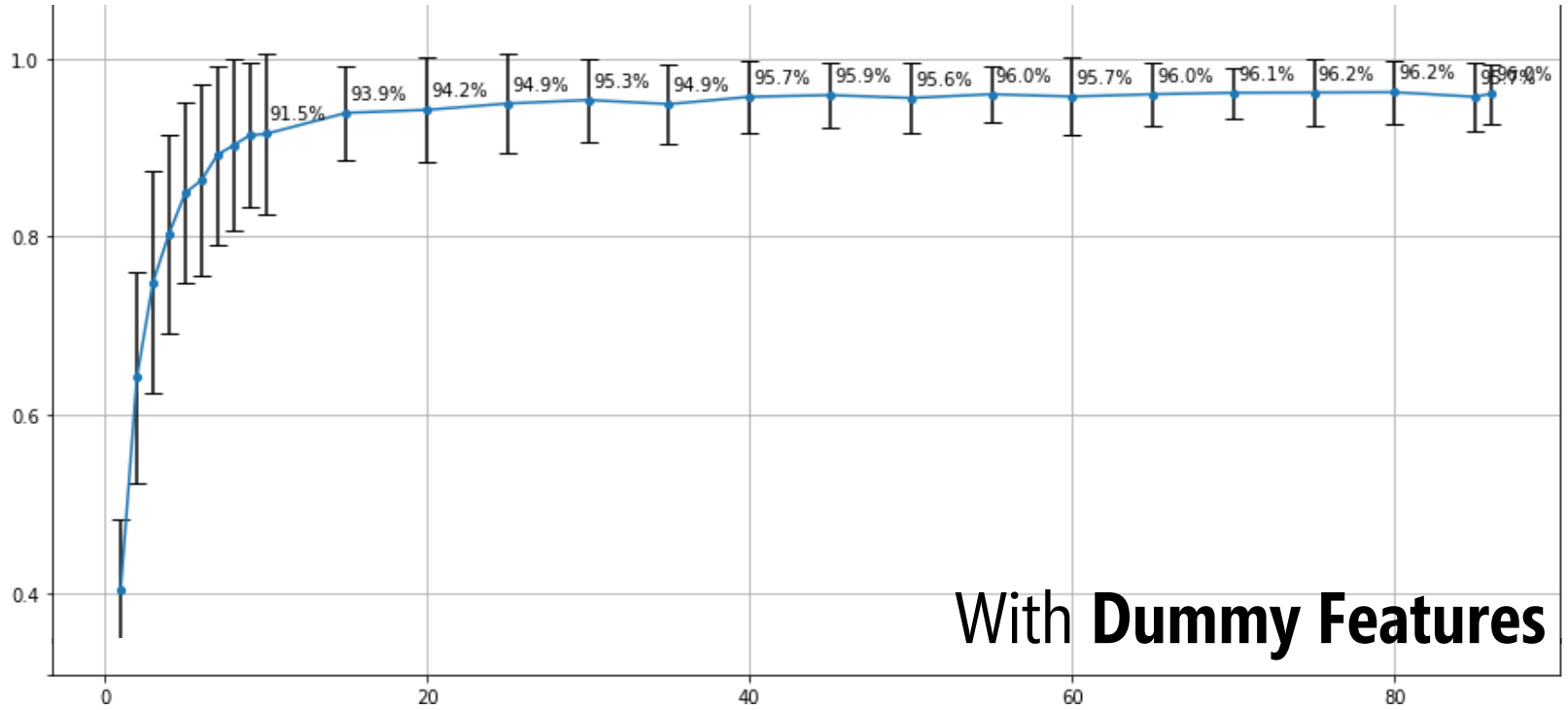
FEATURE SELECTION

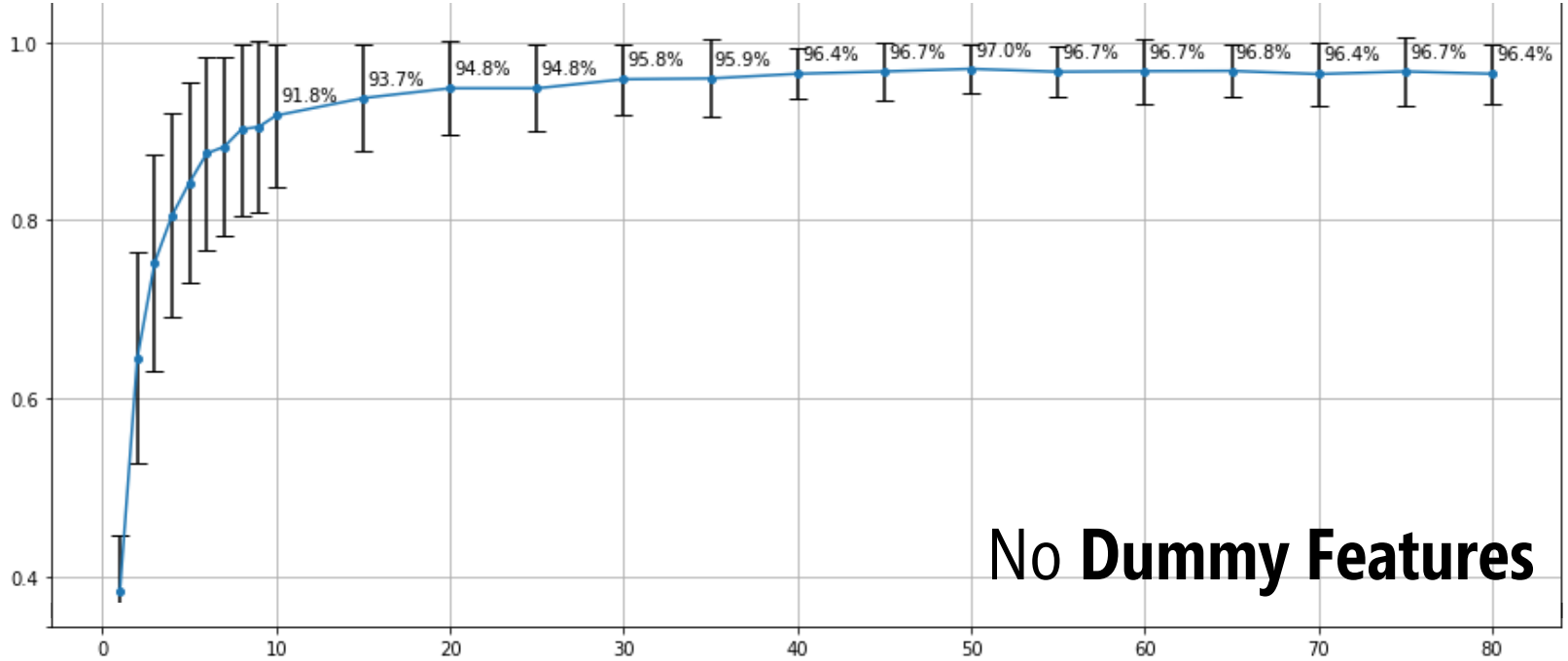
WHAT HAPPENS WHEN WE INJECT 6 DUMMY FEATURES?

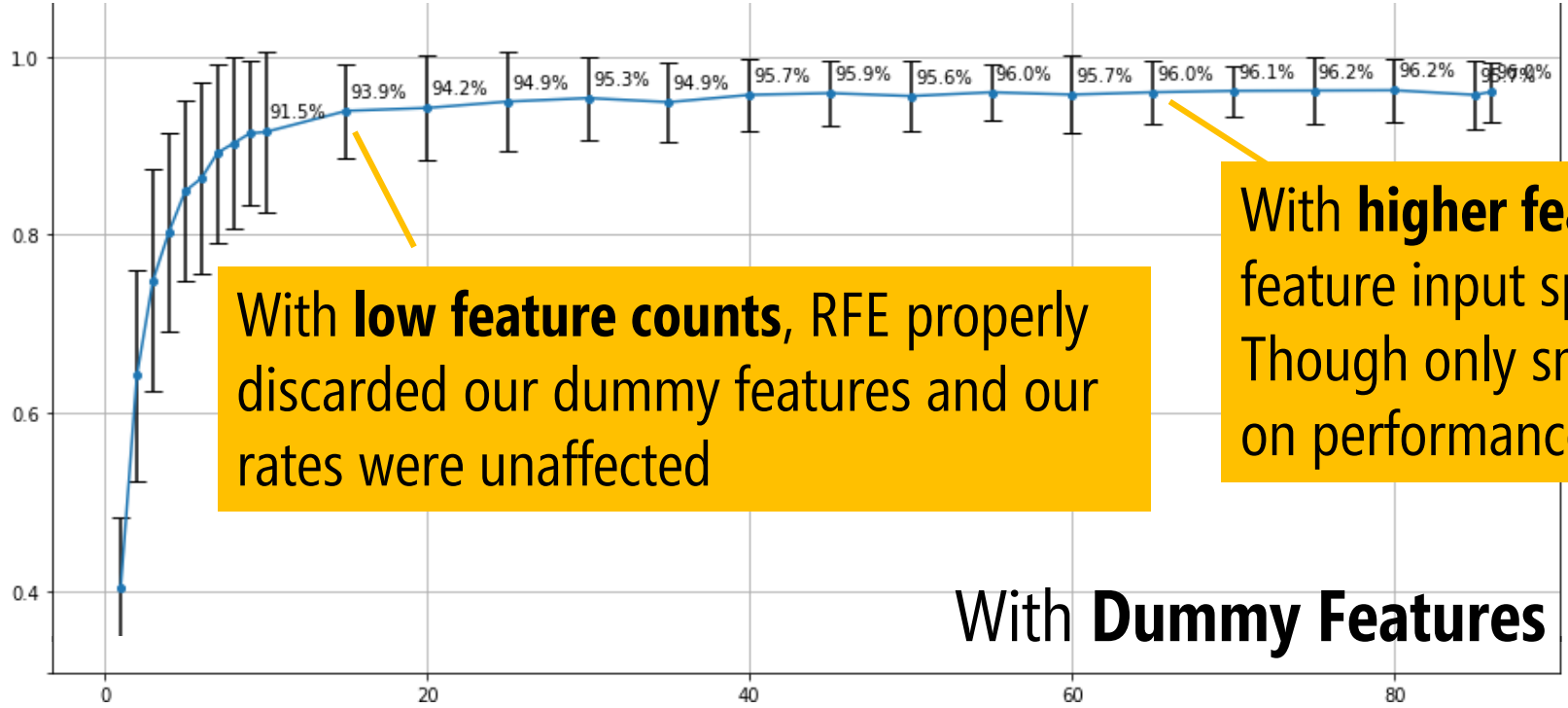
```
# dummy vars to test feature selection
if include_dummy_data:
    features.append(15)
    feature_names.append("dummy_always15")

    features.append(999)
    feature_names.append("dummy_always999")

    val0or40 = 0
    if random.random() > 0.5:
        val0or40 = 40
    features.append(val0or40)
    feature_names.append("dummy_0or40_50%split")

    val0or100 = 0
    if random.random() > 0.9:
        val0or100 = 100
    features.append(val0or100)
    feature_names.append("dummy_0or100_90%split")

    features.append(random.randint(1,101))
    feature_names.append("dummy_randint(1,101)")

    features.append(random.gauss(100, 15))
    feature_names.append("dummy_gauss(100, 15)")
```




1.0
0.8
0.6
0.4
0
10
20
30
40
50
60
70
80
91.8%
93.7%
94.8%
94.8%
95.8%
95.9%
96.4%
96.7%
97.0%
96.7%
96.7%
96.8%
96.4%
96.7%
96.4%
No Dummy Features


1.0
0.8
0.6
0.4
0
20
40
60
80
91.5%
93.9%
94.2%
94.9%
95.3%
94.9%
95.7%
95.9%
95.6%
96.0%
95.7%
96.0%
96.1%
96.2%
96.2%
With Dummy Features





With **low feature counts**, RFE properly discarded our dummy features and our rates were unaffected

With **higher feature counts**, our feature input space got noisier. Though only small negative impact on performance

RFE EXPERIMENTS

DUMMY DATA, INPUT FEATURES=1: 38.4% ACCURACY

For each gesture set, we ran 20 RFE experiments with the feature count set to 1

```
1 FEATURES: OVERALL ACCURACY 40.45% (SD=7.79%)
        FEATURES USED RATES:
        00 y.mean : (27/280) 9.6%
        01 x.max : (26/280) 9.3%
        02 y_p.max : (23/280) 8.2%
        03 y_p.peak_count : (20/280) 7.1%
        04 z_p.peak_avg : (20/280) 7.1%
        05 z.median : (17/280) 6.1%
        06 x.min : (15/280) 5.4%
        07 y_p.min : (14/280) 5.0%
        08 x.var : (12/280) 4.3%
        09 y_p.std : (11/280) 3.9%
        10 x.mean : (11/280) 3.9%
        11 mag_p.top3_freq_coef : (10/280) 3.6%
        12 x_p.std : (9/280) 3.2%
        13 x.std : (6/280) 2.1%
        14 z.min : (5/280) 1.8%
        15 x_p.max : (5/280) 1.8%
```

```
FEATURES DISCARDED RATES
00 length : (280/280) 100.0%
01 dummy_always15 : (280/280) 100.0%
02 dummy_always999 : (280/280) 100.0%
03 dummy_0or40_50%split : (280/280) 100.0%
04 dummy_0or100_90%split : (280/280) 100.0%
05 dummy_randint(1,101) : (280/280) 100.0%
06 dummy_gauss(100, 15) : (280/280) 100.0%
07 y_p.mean : (280/280) 100.0%
08 z_p.mean : (280/280) 100.0%
09 mag_p.median : (280/280) 100.0%
10 x_p.median : (280/280) 100.0%
11 y_p.median : (280/280) 100.0%
12 x.median : (280/280) 100.0%
13 mag_p.std : (280/280) 100.0%
14 y.std : (280/280) 100.0%
15 z.std : (280/280) 100.0%
```

RFE EXPERIMENTS

DUMMY DATA, INPUT FEATURES=1: 38.4% ACCURACY

For each gesture set, we ran 20 RFE experiments with the feature count set to 1

```
1 FEATURES: OVERALL ACCURACY 40.45% (SD=7.79%)
        FEATURES USED RATES:
        00 y.mean : (27/280) 9.6%
        01 x.max : (26/280) 9.3%
        02 y_p.max : (23/280) 8.2%
        03 y_p.peak_count : (20/280) 7.1%
        04 z_p.peak_av
        05 z.median :
        06 x.min : (15
        07 y_p.min : (
        08 x.var : (12/280) 4.3%
        09 y_p.std : (11/280) 3.9%
        10 x.mean : (11/280) 3.9%
        11 mag_p.top3_freq_coef : (10/280) 3.6%
        12 x_p.std : (9/280) 3.2%
        13 x.std : (6/280) 2.1%
        14 z.min : (5/280) 1.8%
        15 x_p.max : (5/280) 1.8%
```

All **six dummy features** discarded every time

```
FEATURES DISCARDED RATES
00 length : (280/280) 100.0%
01 dummy_always15 : (280/280) 100.0%
02 dummy_always999 : (280/280) 100.0%
03 dummy_0or40_50%split : (280/280) 100.0%
04 dummy_0or100_90%split : (280/280) 100.0%
05 dummy_randint(1,101) : (280/280) 100.0%
06 dummy_gauss(100, 15) : (280/280) 100.0%
07 y_p.mean : (280/280) 100.0%
08 z_p.mean : (280/280) 100.0%
09 mag_p.median : (280/280) 100.0%
10 x_p.median : (280/280) 100.0%
11 y_p.median : (280/280) 100.0%
12 x.median : (280/280) 100.0%
13 mag_p.std : (280/280) 100.0%
14 y.std : (280/280) 100.0%
15 z.std : (280/280) 100.0%
```

RFE EXPERIMENTS

DUMMY DATA, INPUT FEATURES=10: 91.5% ACCURACY

For each gesture set, we ran 20 RFE experiments with the feature count set to 10

```
10 FEATURES: OVERALL ACCURACY 91.53% (SD=9.05%)
        FEATURES USED RATES:
        00 y.mean : (173/280) 61.8%
        01 y_p.min : (112/280) 40.0%
        02 z.mean : (105/280) 37.5%
        03 y_p.median : (101/280) 36.1%
        04 x.mean : (93/280) 33.2%
        05 y.median : (90/280) 32.1%
        06 z_p.median : (86/280) 30.7%
        07 z_p.peak_count : (86/280) 30.7%
        08 y_p.peak_count : (83/280) 29.6%
        09 z_p.peak_avg : (79/280) 28.2%
        10 mag.mean : (74/280) 26.4%
        11 x.max : (74/280) 26.4%
        12 x_p.fft_entropy : (74/280) 26.4%
        13 z.median : (72/280) 25.7%
        14 z_p.min : (72/280) 25.7%
        15 x.median : (71/280) 25.4%
```

```
FEATURES DISCARDED RATES
00 dummy_always15 : (280/280) 100.0%
01 dummy_always999 : (280/280) 100.0%
02 dummy_0or100_90%split : (280/280) 100.0%
03 dummy_gauss(100, 15) : (280/280) 100.0%
04 mag_p.std : (280/280) 100.0%
05 y.std : (280/280) 100.0%
06 z.std : (280/280) 100.0%
07 mag_p.var : (280/280) 100.0%
08 mag_p.top3_freq : (280/280) 100.0%
09 x_p.fft_energy : (280/280) 100.0%
10 dummy_randint(1,101) : (279/280) 99.6%
11 z.var : (279/280) 99.6%
12 mag_p.top1_freq : (279/280) 99.6%
13 mag_p.fft_entropy : (279/280) 99.6%
14 mag_p.fft_energy : (279/280) 99.6%
15 y_p.fft_energy : (279/280) 99.6%
16 dummy_0or40_50%split : (278/280) 99.3%
```

RFE EXPERIMENTS

DUMMY DATA, INPUT FEATURES=60: 95.7% ACCURACY

For each gesture set, we ran 20 RFE experiments with the feature count set to 10

```
60 FEATURES: OVERALL ACCURACY 95.71% (SD=4.39%)
        FEATURES USED RATES:
        00 y.median : (280/280) 100.0%
        01 mag_p.peak_count : (280/280) 100.0%
        02 y.mean : (279/280) 99.6%
        03 y_p.min : (279/280) 99.6%
        04 y_p.std : (278/280) 99.3%
        05 y_p.median : (276/280) 98.6%
        06 z_p.peak_avg : (275/280) 98.2%
        07 z_p.median : (273/280) 97.5%
        08 y.min : (271/280) 96.8%
        09 x.median : (266/280) 95.0%
        10 x.min : (264/280) 94.3%
        11 length : (263/280) 93.9%
        12 y_p.max : (263/280) 93.9%
        13 z_p.fft_entropy : (263/280) 93.9%
        14 z_p.min : (261/280) 93.2%
        15 y_p.peak_count : (261/280) 93.2%
        16 y_p.peak_avg : (260/280) 92.9%
        17 x.mean : (259/280) 92.5%
```

```
FEATURES DISCARDED RATES
00 dummy_always15 : (280/280) 100.0%
01 dummy_always999 : (280/280) 100.0%
02 mag.var : (239/280) 85.4%
03 mag_p.fft_energy : (230/280) 82.1%
04 mag.std : (226/280) 80.7%
05 x_p.fft_energy : (209/280) 74.6%
06 z.var : (204/280) 72.9%
07 dummy_0or100_90%split : (191/280) 68.2%
08 mag_p.std : (183/280) 65.4%
09 mag_p.var : (183/280) 65.4%
10 mag_p.zero_crossings : (165/280) 58.9%
11 z_p.fft_energy : (165/280) 58.9%
12 y.var : (162/280) 57.9%
13 x_p.var : (158/280) 56.4%
14 y_p.max_freq : (154/280) 55.0%
15 z_p.var : (143/280) 51.1%
16 x.var : (140/280) 50.0%
17 z_p.zero_crossings : (137/280) 48.9%
```

60 FEATURES: OVERALL ACCURACY 95.71% (SD=4.39%)

FEATURES USED RATES:

00 y.median : (280/280) 100.0%
01 mag_p.peak_count : (280/280) 100.0%
02 y.mean : (279/280) 99.6%
03 y_p.min : (279/280) 99.6%
04 y_p.std : (278/280) 99.3%
05 y_p.median : (276/280) 98.6%
06 z_p.peak_avg : (275/280) 98.2%
07 z_p.median : (273/280) 97.5%
08 y.min : (271/280) 96.8%
09 x.median : (266/280) 95.0%
10 x.min : (264/280) 94.3%
11 length : (263/280) 93.9%
12 y_p.max : (263/280) 93.9%
13 z_p.fft_entropy : (263/280) 93.9%
14 z_p.min : (261/280) 93.2%
15 y_p.peak_count : (261/280) 93.2%
16 y_p.peak_avg : (260/280) 92.9%
17 x.mean : (259/280) 92.5%
18 z_p.peak_count : (255/280) 91.1%
19 y_p.fft_entropy : (252/280) 90.0%
20 y.max : (251/280) 89.6%
21 x_p.fft_entropy : (251/280) 89.6%
22 x.max : (248/280) 88.6%
23 x_p.peak_count : (247/280) 88.2%
24 mag.min : (245/280) 87.5%
25 x.std : (244/280) 87.1%
26 y_p.mean : (243/280) 86.8%
27 z.mean : (242/280) 86.4%
28 x_p.median : (238/280) 85.0%
29 mag.median : (238/280) 85.0%
30 mag_p.median : (236/280) 84.3%
31 z_p.std : (235/280) 83.9%
32 x_p.peak_avg : (235/280) 83.9%
33 z.min : (234/280) 83.6%
34 dummy_randint(1,101) : (232/280) 82.9%
35 z_p.max : (231/280) 82.5%
36 z.median : (230/280) 82.1%
37 mag.mean : (228/280) 81.4%
38 x_p.max : (224/280) 80.0%
39 mag_p.min : (224/280) 80.0%
40 mag_p.peak_avg : (224/280) 80.0%
41 z.max : (221/280) 78.9%
42 dummy_0or40_50%split : (219/280) 78.2%
43 mag_p.fft_entropy : (219/280) 78.2%
44 dummy_gauss(100, 15) : (217/280) 77.5%

FEATURES DISCARDED RATES

00 dummy_always15 : (280/280) 100.0%
01 dummy_always999 : (280/280) 100.0%
02 mag.var : (239/280) 85.4%
03 mag_p.fft_energy : (230/280) 82.1%
04 mag.std : (226/280) 80.7%
05 x_p.fft_energy : (209/280) 74.6%
06 z.var : (204/280) 72.9%
07 dummy_0or100_90%split : (191/280) 68.2%
08 mag_p.std : (183/280) 65.4%
09 mag_p.var : (183/280) 65.4%
10 mag_p.zero_crossings : (165/280) 58.9%
11 z_p.fft_energy : (165/280) 58.9%
12 y.var : (162/280) 57.9%
13 x_p.var : (158/280) 56.4%
14 y_p.max_freq : (154/280) 55.0%
15 z_p.var : (143/280) 51.1%
16 x.var : (140/280) 50.0%
17 z_p.zero_crossings : (137/280) 48.9%
18 y_p.fft_energy : (137/280) 48.9%
19 x_p.max_freq : (128/280) 45.7%
20 y_p.zero_crossings : (121/280) 43.2%
21 x_p.zero_crossings : (120/280) 42.9%
22 mag_p.max : (114/280) 40.7%
23 mag_p.mean : (113/280) 40.4%
24 mag_p.top1_freq_coef : (111/280) 39.6%
25 mag.max : (109/280) 38.9%
26 z_p.max_freq : (99/280) 35.4%
27 z.std : (98/280) 35.0%
28 mag_p.top2_freq : (97/280) 34.6%
29 mag_p.top1_freq : (97/280) 34.6%
30 mag_p.top3_freq_coef : (96/280) 34.3%
31 z_p.mean : (90/280) 32.1%
32 mag_p.top3_freq : (90/280) 32.1%
33 mag_p.top2_freq_coef : (87/280) 31.1%
34 x_p.mean : (87/280) 31.1%
35 y_p.var : (80/280) 28.6%
36 mag_p.max_freq : (78/280) 27.9%
37 y.std : (73/280) 26.1%
38 x_p.std : (73/280) 26.1%
39 x_p.min : (67/280) 23.9%
40 dummy_gauss(100, 15) : (63/280) 22.5%
41 mag_p.fft_entropy : (61/280) 21.8%
42 dummy_0or40_50%split : (61/280) 21.8%
43 z.max : (59/280) 21.1%
44 mag_p.peak_avg : (56/280) 20.0%
45 x_p.max : (56/280) 20.0%

Parameter Tuning

(see Jupyter Notebook)