

# Pattern & Anomaly Detection Lab

## Experiment 11

### Submitted By:

Dhruv Singhal

500075346

R177219074

AIML B3

### Submitted To:

Dr. Gopal Phartiyal

Asst. Professor

SOCS

UPES

# NOTE:

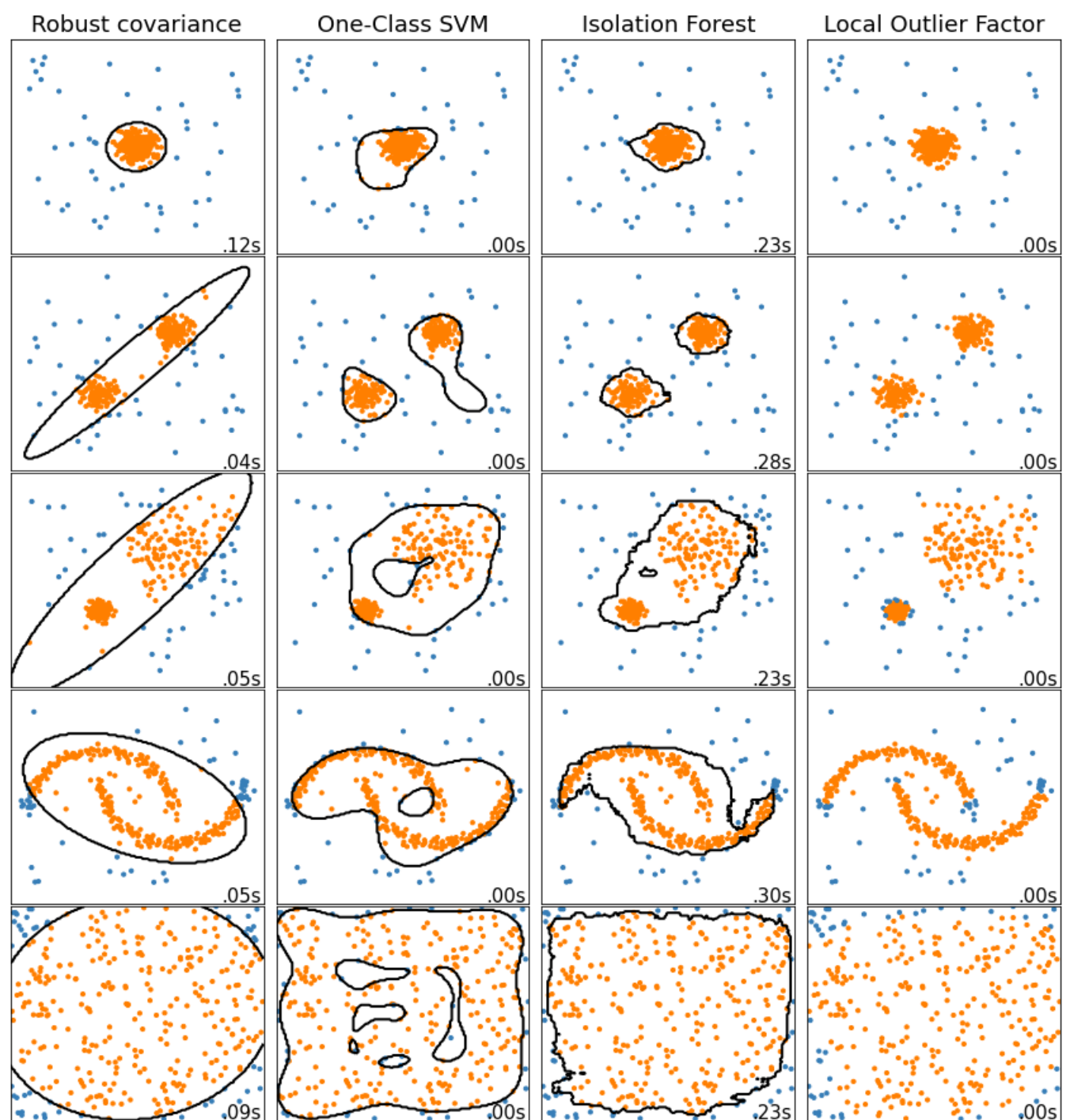
```
1 """
2 =====
3 Comparing anomaly detection algorithms for outlier detection on toy datasets
4 =====
5
6 This example shows characteristics of different anomaly detection algorithms
7 on 2D datasets. Datasets contain one or two modes (regions of high density)
8 to illustrate the ability of algorithms to cope with multimodal data.
9
10 For each dataset, 15% of samples are generated as random uniform noise. This
11 proportion is the value given to the nu parameter of the OneClassSVM and the
12 contamination parameter of the other outlier detection algorithms.
13 Decision boundaries between inliers and outliers are displayed in black
14 except for Local Outlier Factor (LOF) as it has no predict method to be applied
15 on new data when it is used for outlier detection.
16
17 The :class:`~sklearn.svm.OneClassSVM` is known to be sensitive to outliers and
18 thus does not perform very well for outlier detection. This estimator is best
19 suited for novelty detection when the training set is not contaminated by
20 outliers. That said, outlier detection in high-dimension, or without any
21 assumptions on the distribution of the inlying data is very challenging, and a
22 One-class SVM might give useful results in these situations depending on the
23 value of its hyperparameters.
24
25 The :class:`~sklearn.linear_model.SGDOneClassSVM` is an implementation of the
26 One-Class SVM based on stochastic gradient descent (SGD). Combined with kernel
27 approximation, this estimator can be used to approximate the solution
28 of a kernelized :class:`~sklearn.svm.OneClassSVM`. We note that, although not
29 identical, the decision boundaries of the
30 :class:`~sklearn.linear_model.SGDOneClassSVM` and the ones of
31 :class:`~sklearn.svm.OneClassSVM` are very similar. The main advantage of using
32 :class:`~sklearn.linear_model.SGDOneClassSVM` is that it scales linearly with
33 the number of samples.
34
35 :class:`~sklearn.covariance.LowRankCovariance` assumes the data is Gaussian and
36 learns an ellipse. It thus degrades when the data is not unimodal. Notice
37 however that this estimator is robust to outliers.
38
39 :class:`~sklearn.ensemble.IsolationForest` and
40 :class:`~sklearn.neighbors.LocalOutlierFactor` seem to perform reasonably well
41 for multi-modal data sets. The advantage of
42 :class:`~sklearn.neighbors.LocalOutlierFactor` over the other estimators is
43 shown for the third data set, where the two modes have different densities.
44 This advantage is explained by the local aspect of LOF, meaning that it only
45 compares the score of abnormality of one sample with the scores of its
46 neighbors.
47
48 Finally, for the last data set, it is hard to say that one sample is more
49 abnormal than another sample as they are uniformly distributed in a
50 hypercube. Except for the :class:`~sklearn.svm.OneClassSVM` which overfits a
51 little, all estimators present decent solutions for this situation. In such a
52 case, it would be wise to look more closely at the scores of abnormality of
53 the samples as a good estimator should assign similar scores to all the
54 samples.
55
56 While these examples give some intuition about the algorithms, this
57 intuition might not apply to very high dimensional data.
58
59 Finally, note that parameters of the models have been here handpicked but
60 that in practice they need to be adjusted. In the absence of labelled data,
61 the problem is completely unsupervised so model selection can be a challenge.
62 """
63
```

# CODE:

```
65 import time
66
67 import numpy as np
68 import matplotlib
69 import matplotlib.pyplot as plt
70
71 from sklearn import svm
72 from sklearn.datasets import make_moons, make_blobs
73 from sklearn.covariance import EllipticEnvelope
74 from sklearn.ensemble import IsolationForest
75 from sklearn.neighbors import LocalOutlierFactor
76 #from sklearn.linear_model import SGDOneClassSVM
77 #from sklearn.kernel_approximation import Nystroem
78 #from sklearn.pipeline import make_pipeline
79
80 matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
81
82 # Example settings
83 n_samples = 300
84 outliers_fraction = 0.15
85 n_outliers = int(outliers_fraction * n_samples)
86 n_inliers = n_samples - n_outliers
87
88 # define outlier/anomaly detection methods to be compared.
89 # the SGDOneClassSVM must be used in a pipeline with a kernel approximation
90 # to give similar results to the OneClassSVM
91 anomaly_algorithms = [
92     ("Robust covariance", EllipticEnvelope(contamination=outliers_fraction)),
93     ("One-class SVM", svm.OneClassSVM(nu=outliers_fraction, kernel="rbf",
94                                     gamma=0.1)),
95     # ("One-class SVM (SGD)", make_pipeline(
96     #     Nystroem(gamma=0.1, random_state=42, n_components=150),
97     #     SGDOneClassSVM(nu=outliers_fraction, shuffle=True,
98     #                   fit_intercept=True, random_state=42, tol=1e-6)
99     # )),
100     ("Isolation Forest", IsolationForest(contamination=outliers_fraction,
101                                         random_state=42)),
102     ("Local Outlier Factor", LocalOutlierFactor(
103         n_neighbors=35, contamination=outliers_fraction))
104
105 # Define datasets
106 blobs_params = dict(random_state=0, n_samples=n_inliers, n_features=2)
107 datasets = [
108     make_blobs(centers=[[0, 0], [0, 0]], cluster_std=0.5,
109               **blobs_params)[0],
110     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[0.5, 0.5],
111               **blobs_params)[0],
112     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[1.5, .3],
113               **blobs_params)[0],
114     4. * (make_moons(n_samples=n_samples, noise=.05, random_state=0)[0] -
115          np.array([0.5, 0.25])),
116     14. * (np.random.RandomState(42).rand(n_samples, 2) - 0.5)]
117
118 # Compare given classifiers under given settings
119 xx, yy = np.meshgrid(np.linspace(-7, 7, 150),
120                     np.linspace(-7, 7, 150))
121
122 plt.figure(figsize=(len(anomaly_algorithms) * 2 + 4, 12.5))
123 plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
124                   hspace=.01)
125
126 plot_num = 1
127 rng = np.random.RandomState(42)
```

```
128 for i_dataset, X in enumerate(datasets):
129     # Add outliers
130     X = np.concatenate([X, rng.uniform(low=-6, high=6, size=(n_outliers, 2))],
131                       axis=0)
132
133 for name, algorithm in anomaly_algorithms:
134     t0 = time.time()
135     algorithm.fit(X)
136     t1 = time.time()
137     plt.subplot(len(datasets), len(anomaly_algorithms), plot_num)
138     if i_dataset == 0:
139         plt.title(name, size=18)
140
141     # fit the data and tag outliers
142     if name == "Local Outlier Factor":
143         y_pred = algorithm.fit_predict(X)
144     else:
145         y_pred = algorithm.fit(X).predict(X)
146
147     # plot the levels lines and the points
148     if name != "Local Outlier Factor": # LOF does not implement predict
149         Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
150         Z = Z.reshape(xx.shape)
151         plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
152
153     colors = np.array(['#377eb8', '#ff7f00'])
154     plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[(y_pred + 1) // 2])
155
156     plt.xlim(-7, 7)
157     plt.ylim(-7, 7)
158     plt.xticks(())
159     plt.yticks(())
160     plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
161             transform=plt.gca().transAxes, size=15,
162             horizontalalignment='right')
163     plot_num += 1
164
165 plt.show()
```

# OUTPUT:

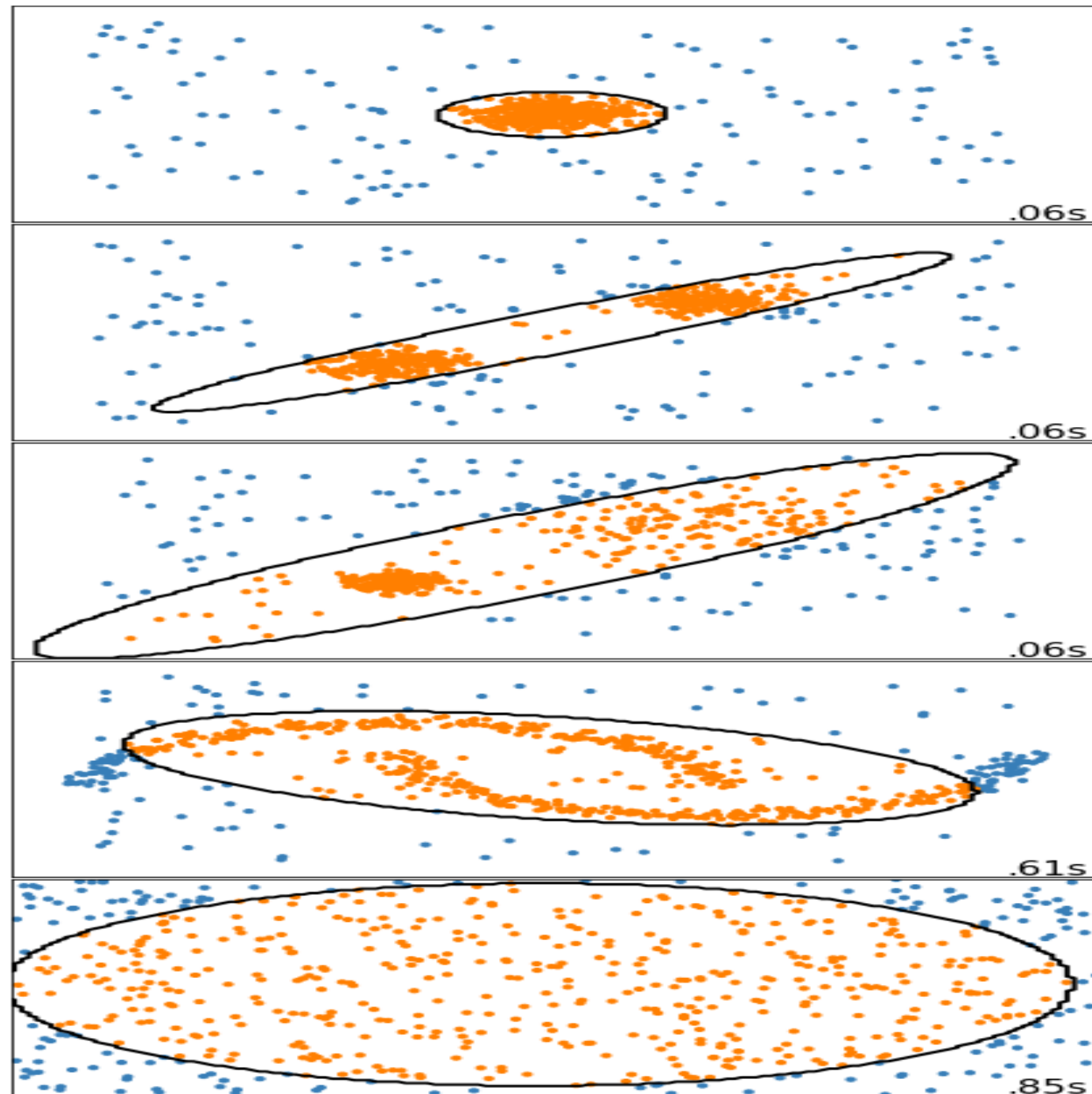


# CODE: (Robust Covariance)

```
168 """
169 matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
170 n_samples = 500
171 outliers_fraction = 0.25
172 n_outliers = int(outliers_fraction * n_samples)
173 n_inliers = n_samples - n_outliers
174 anomaly_algorithms = [
175     ("Robust covariance", EllipticEnvelope(store_precision=False, contamination=outliers_fraction, random_state=42))
176 ]
177 blobs_params = dict(random_state=0, n_samples=n_inliers, n_features=2)
178 datasets = [
179     make_blobs(centers=[[0, 0], [0, 0]], cluster_std=0.5,
180               **blobs_params)[0],
181     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[0.5, 0.5],
182               **blobs_params)[0],
183     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[1.5, .3],
184               **blobs_params)[0],
185     4. * (make_moons(n_samples=n_samples, noise=.05, random_state=0)[0] -
186           np.array([0.5, 0.25])),
187     14. * (np.random.RandomState(42).rand(n_samples, 2) - 0.5)]
188 xx, yy = np.meshgrid(np.linspace(-7, 7, 150),
189                     np.linspace(-7, 7, 150))
190
191 plt.figure(figsize=(len(anomaly_algorithms) * 2 + 4, 12.5))
192 plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
193                   hspace=.01)
194
195 plot_num = 1
196 rng = np.random.RandomState(42)
197
198 for i_dataset, X in enumerate(datasets):
199     # Add outliers
200     X = np.concatenate([X, rng.uniform(low=-6, high=6, size=(n_outliers, 2))],
201                       axis=0)
202
203     for name, algorithm in anomaly_algorithms:
204         t0 = time.time()
205         algorithm.fit(X)
206         t1 = time.time()
207         plt.subplot(len(datasets), len(anomaly_algorithms), plot_num)
208         if i_dataset == 0:
209             plt.title(name, size=18)
210
211         # fit the data and tag outliers
212         if name == "Local Outlier Factor":
213             y_pred = algorithm.fit_predict(X)
214         else:
215             y_pred = algorithm.fit(X).predict(X)
216         # plot the levels lines and the points
217         if name != "Local Outlier Factor": # LOF does not implement predict
218             Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
219             Z = Z.reshape(xx.shape)
220             plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
221
222         colors = np.array(['#377eb8', '#ff7f00'])
223         plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[(y_pred + 1) // 2])
224
225         plt.xlim(-7, 7)
226         plt.ylim(-7, 7)
227         plt.xticks(())
228         plt.yticks(())
229         plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
230               transform=plt.gca().transAxes, size=15,
231               horizontalalignment='right')
232         plot_num += 1
233     plt.show()
```



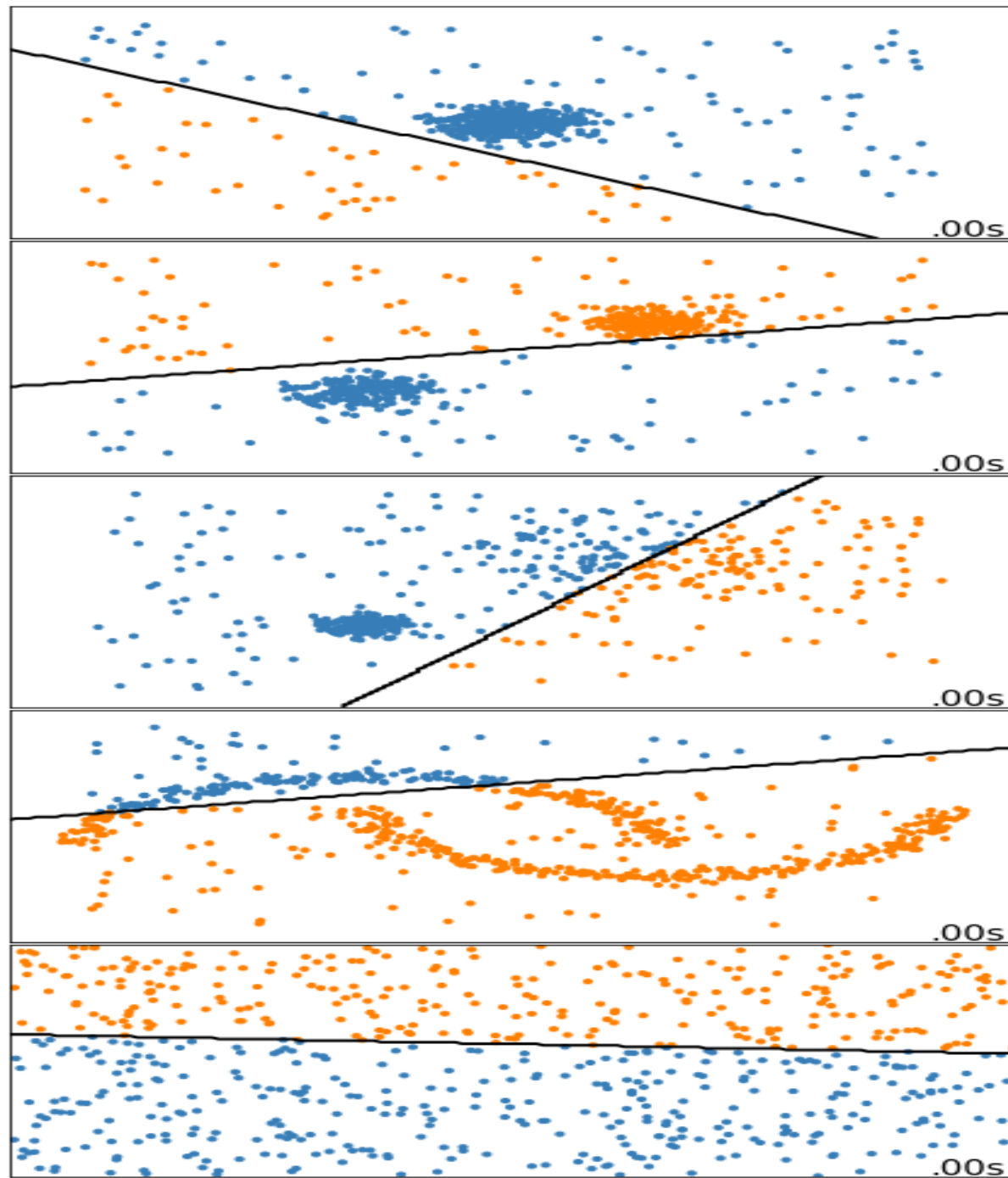
# Robust covariance



# CODE: (One-Class SVM)

```
233 #%%
234 #%%
235 matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
236 n_samples = 500
237 outliers_fraction = 0.25
238 n_outliers = int(outliers_fraction * n_samples)
239 n_inliers = n_samples - n_outliers
240
241 anomaly_algorithms = [
242     ("One-Class SVM", svm.OneClassSVM(nu=outliers_fraction, kernel="linear", degree=4,
243     gamma=0.1)), ]
244 blobs_params = dict(random_state=0, n_samples=n_inliers, n_features=2)
245 datasets = [
246     make_blobs(centers=[[0, 0], [0, 0]], cluster_std=0.5,
247     **blobs_params)[0],
248     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[0.5, 0.5],
249     **blobs_params)[0],
250     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[1.5, .3],
251     **blobs_params)[0],
252     4. * (make_moons(n_samples=n_samples, noise=.05, random_state=0)[0] -
253     np.array([0.5, 0.25])),
254     14. * (np.random.RandomState(42).rand(n_samples, 2) - 0.5)]
255 # Compare given classifiers under given settings
256 xx, yy = np.meshgrid(np.linspace(-7, 7, 150),
257     np.linspace(-7, 7, 150))
258 plt.figure(figsize=(len(anomaly_algorithms) * 2 + 4, 12.5))
259 plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
260     hspace=.01)
261 plot_num = 1
262 rng = np.random.RandomState(42)
263
264 for i_dataset, X in enumerate(datasets):
265     # Add outliers
266     X = np.concatenate([X, rng.uniform(low=-6, high=6, size=(n_outliers, 2))],
267     axis=0)
268     for name, algorithm in anomaly_algorithms:
269         t0 = time.time()
270         algorithm.fit(X)
271         t1 = time.time()
272         plt.subplot(len(datasets), len(anomaly_algorithms), plot_num)
273         if i_dataset == 0:
274             plt.title(name, size=18)
275         # fit the data and tag outliers
276         if name == "Local Outlier Factor":
277             y_pred = algorithm.fit_predict(X)
278         else:
279             y_pred = algorithm.fit(X).predict(X)
280         # plot the levels lines and the points
281         if name != "Local Outlier Factor": # LOF does not implement predict
282             Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
283             Z = Z.reshape(xx.shape)
284             plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
285
286         colors = np.array(['#377eb8', '#ff7f00'])
287         plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[(y_pred + 1) // 2])
288         plt.xlim(-7, 7)
289         plt.ylim(-7, 7)
290         plt.xticks(())
291         plt.yticks(())
292         plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
293             transform=plt.gca().transAxes, size=15,
294             horizontalalignment='right')
295         plot_num += 1
296 plt.show()
297
```

One-Class SVM

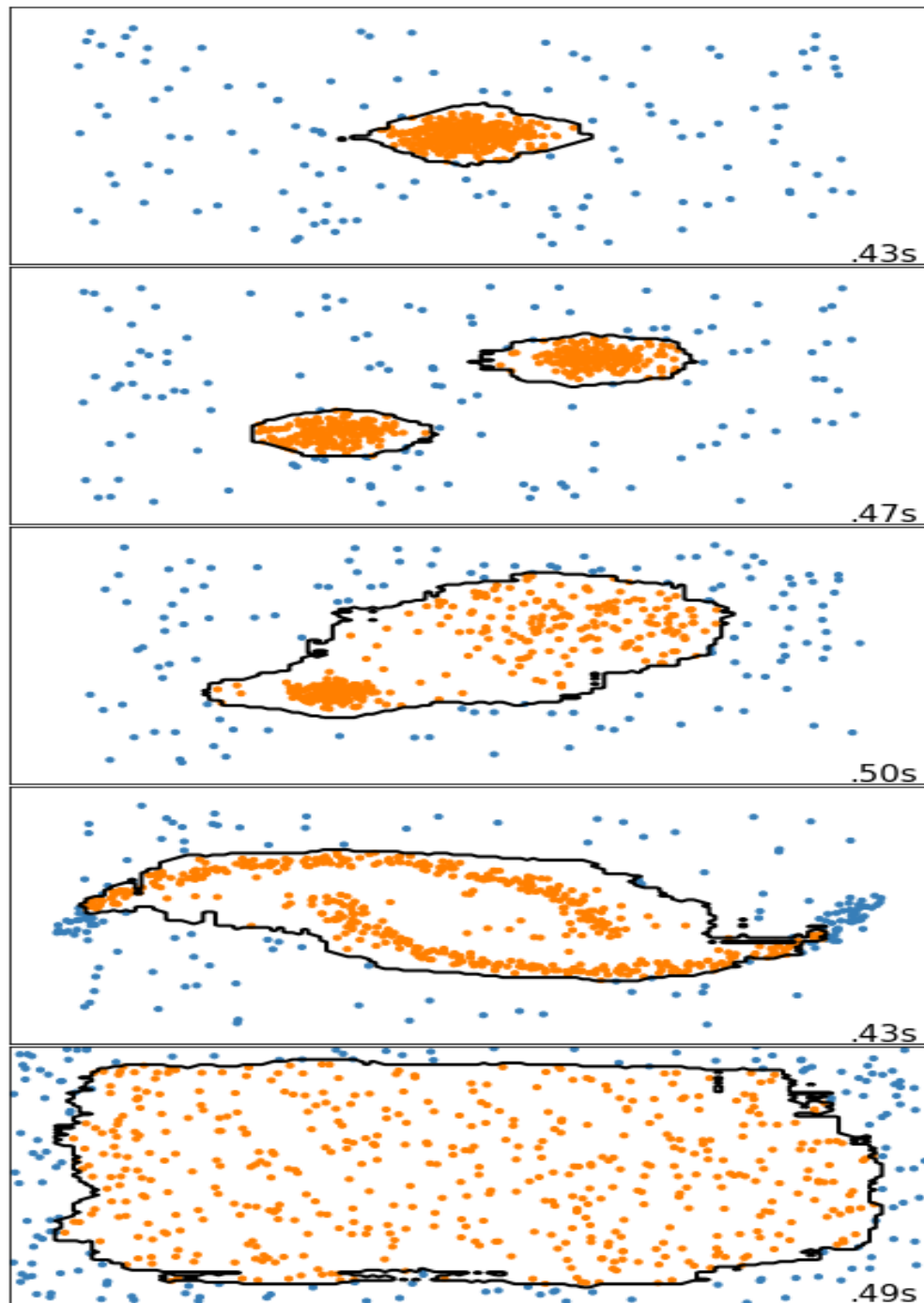




# CODE: (Isolation Forest)

```
298 #%%
299 matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
300 n_samples = 500
301 outliers_fraction = 0.25
302 n_outliers = int(outliers_fraction * n_samples)
303 n_inliers = n_samples - n_outliers
304 anomaly_algorithms = [ ("Isolation Forest", IsolationForest(contamination=outliers_fraction,bootstrap=True,
305                                                             random_state=42))]
306 blobs_params = dict(random_state=0, n_samples=n_inliers, n_features=2)
307 datasets = [
308     make_blobs(centers=[[0, 0], [0, 0]], cluster_std=0.5,
309               **blobs_params)[0],
310     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[0.5, 0.5],
311               **blobs_params)[0],
312     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[1.5, .3],
313               **blobs_params)[0],
314     4. * (make_moons(n_samples=n_samples, noise=.05, random_state=0)[0] -
315           np.array([0.5, 0.25])),
316     14. * (np.random.RandomState(42).rand(n_samples, 2) - 0.5)]
317 # Compare given classifiers under given settings
318 xx, yy = np.meshgrid(np.linspace(-7, 7, 150),
319                     np.linspace(-7, 7, 150))
320
321 plt.figure(figsize=(len(anomaly_algorithms) * 2 + 4, 12.5))
322 plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
323                   hspace=.01)
324 plot_num = 1
325 rng = np.random.RandomState(42)
326
327 for i_dataset, X in enumerate(datasets):
328     # Add outliers
329     X = np.concatenate([X, rng.uniform(low=-6, high=6, size=(n_outliers, 2))],
330                       axis=0)
331
332     for name, algorithm in anomaly_algorithms:
333         t0 = time.time()
334         algorithm.fit(X)
335         t1 = time.time()
336         plt.subplot(len(datasets), len(anomaly_algorithms), plot_num)
337         if i_dataset == 0:
338             plt.title(name, size=18)
339
340         # fit the data and tag outliers
341         if name == "Local Outlier Factor":
342             y_pred = algorithm.fit_predict(X)
343         else:
344             y_pred = algorithm.fit(X).predict(X)
345         # plot the levels lines and the points
346         if name != "Local Outlier Factor": # LOF does not implement predict
347             Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
348             Z = Z.reshape(xx.shape)
349             plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
350
351         colors = np.array(['#377eb8', '#ff7f00'])
352         plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[(y_pred + 1) // 2])
353         plt.xlim(-7, 7)
354         plt.ylim(-7, 7)
355         plt.xticks(())
356         plt.yticks(())
357         plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
358               transform=plt.gca().transAxes, size=15,
359               horizontalalignment='right')
360         plot_num += 1
361 plt.show()
362 #%%
```

# Isolation Forest



# CODE: (Local Outlier Factor)

```
363 #%%
364 matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
365 n_samples = 500
366 outliers_fraction = 0.25
367 n_outliers = int(outliers_fraction * n_samples)
368 n_inliers = n_samples - n_outliers
369 anomaly_algorithms = [("Local Outlier Factor", LocalOutlierFactor(
370     n_neighbors=5, algorithm='kd_tree', contamination=outliers_fraction))]
371 blobs_params = dict(random_state=0, n_samples=n_inliers, n_features=2)
372 datasets = [
373     make_blobs(centers=[[0, 0], [0, 0]], cluster_std=0.5,
374         **blobs_params)[0],
375     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[0.5, 0.5],
376         **blobs_params)[0],
377     make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[1.5, .3],
378         **blobs_params)[0],
379     4. * (make_moons(n_samples=n_samples, noise=.05, random_state=0)[0] -
380         np.array([0.5, 0.25])),
381     14. * (np.random.RandomState(42).rand(n_samples, 2) - 0.5)]
382 # Compare given classifiers under given settings
383 xx, yy = np.meshgrid(np.linspace(-7, 7, 150),
384     np.linspace(-7, 7, 150))
385 plt.figure(figsize=(len(anomaly_algorithms) * 2 + 4, 12.5))
386 plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
387     hspace=.01)
388
389 plot_num = 1
390 rng = np.random.RandomState(42)
391
392 for i_dataset, X in enumerate(datasets):
393     # Add outliers
394     X = np.concatenate([X, rng.uniform(low=-6, high=6, size=(n_outliers, 2))],
395         axis=0)
396
397     for name, algorithm in anomaly_algorithms:
398         t0 = time.time()
399         algorithm.fit(X)
400         t1 = time.time()
401         plt.subplot(len(datasets), len(anomaly_algorithms), plot_num)
402         if i_dataset == 0:
403             plt.title(name, size=18)
404
405         # fit the data and tag outliers
406         if name == "Local Outlier Factor":
407             y_pred = algorithm.fit_predict(X)
408         else:
409             y_pred = algorithm.fit(X).predict(X)
410
411         # plot the levels lines and the points
412         if name != "Local Outlier Factor": # LOF does not implement predict
413             Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
414             Z = Z.reshape(xx.shape)
415             plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')
416             colors = np.array(['#377eb8', '#ff7f00'])
417             plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[(y_pred + 1) // 2])
418             plt.xlim(-7, 7)
419             plt.ylim(-7, 7)
420             plt.xticks(())
421             plt.yticks(())
422             plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
423                 transform=plt.gca().transAxes, size=15,
424                 horizontalalignment='right')
425         plot_num += 1
426 plt.show()
```

Local Outlier Factor

