W4995 Applied Machine Learning

Working with Imbalanced Data

03/20/17

Andreas Müller

Recap on imbalanced data

Two sources of imbalance

- Asymmetric cost
- Asymmetric data

Why do we care?

- Why should cost be symmetric?
- Detect rare events

Changing Thresholds

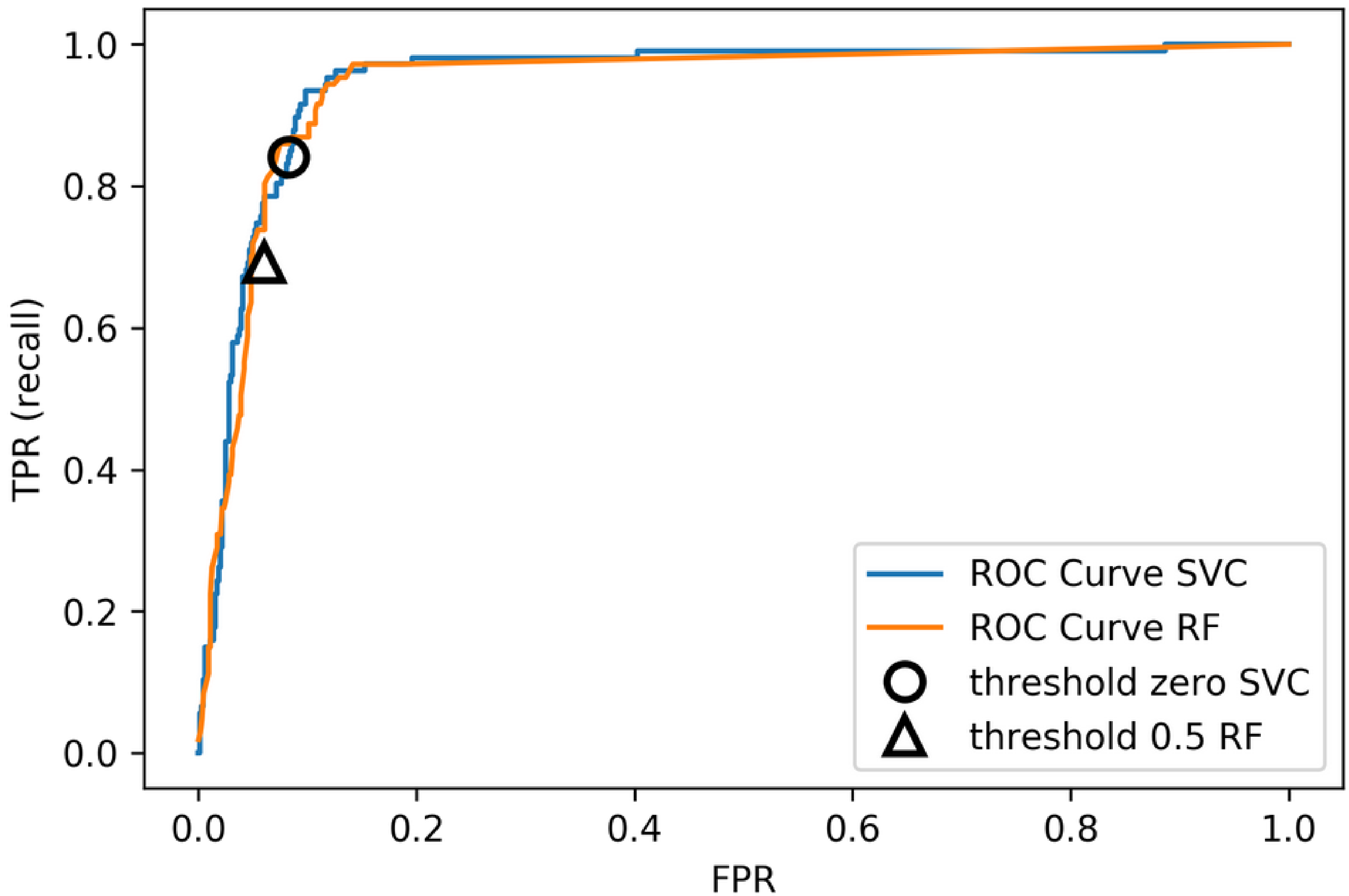
```
# logistic regresson on breast cancer, but change threshold:
data = load_breast_cancer()

X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, stratify=data.target, random_state=0)

lr = LogisticRegression().fit(X_train, y_train)
y_pred = lr.predict(X_test)

print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.91      0.92      0.92        53
          1       0.96      0.94      0.95        90

avg / total       0.94      0.94      0.94       143
```

```
y_pred = lr.predict_proba(X_test)[:, 1] > .85

print(classification_report(y_test, y_pred))
```

```
             precision    recall  f1-score   support

          0       0.84      1.00      0.91        53
          1       1.00      0.89      0.94        90

avg / total       0.94      0.93      0.93       143
```

Roc Curve



1.0
0.8
0.6
0.4
0.2
0.0
TPR (recall)
0.0
0.2
0.4
0.6
0.8
1.0
FPR
ROC Curve SVC
ROC Curve RF
threshold zero SVC
threshold 0.5 RF

Remedies for the model

Mammography data

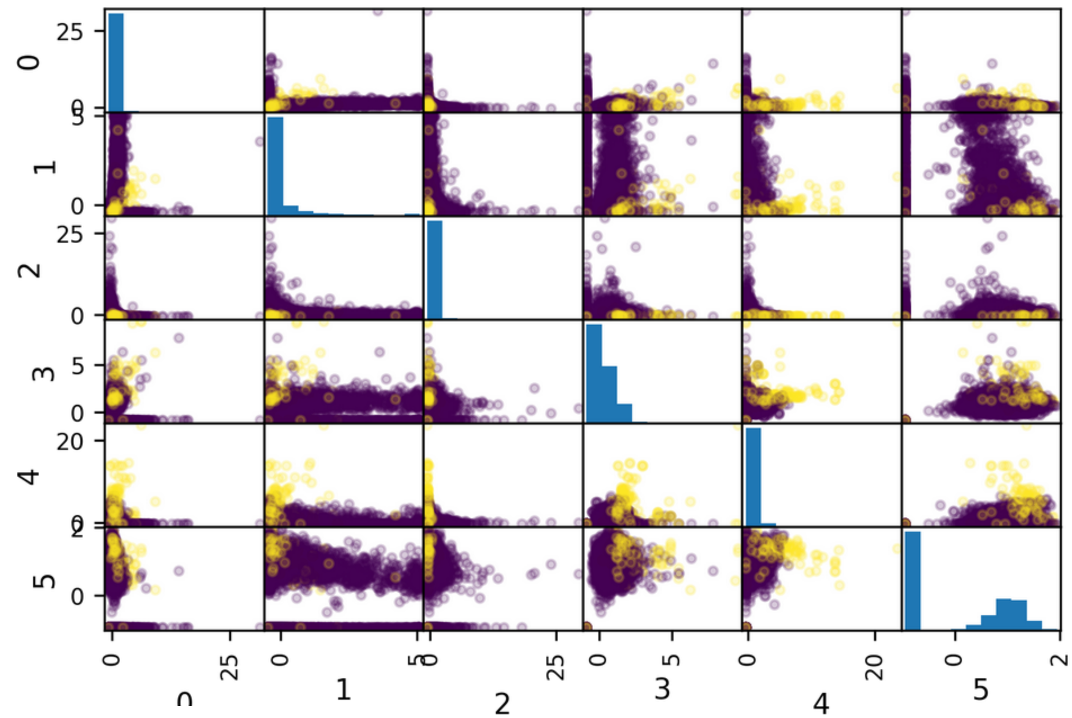
```
import openml
# mammography dataset https://www.openml.org/d/310
data = openml.datasets.get_dataset(310)
X, y = data.get_data(target=data.default_target_attribute)
```

```
X.shape
```

```
(11183, 6)
```

```
np.bincount(y)
```

```
array([10923,   260])
```



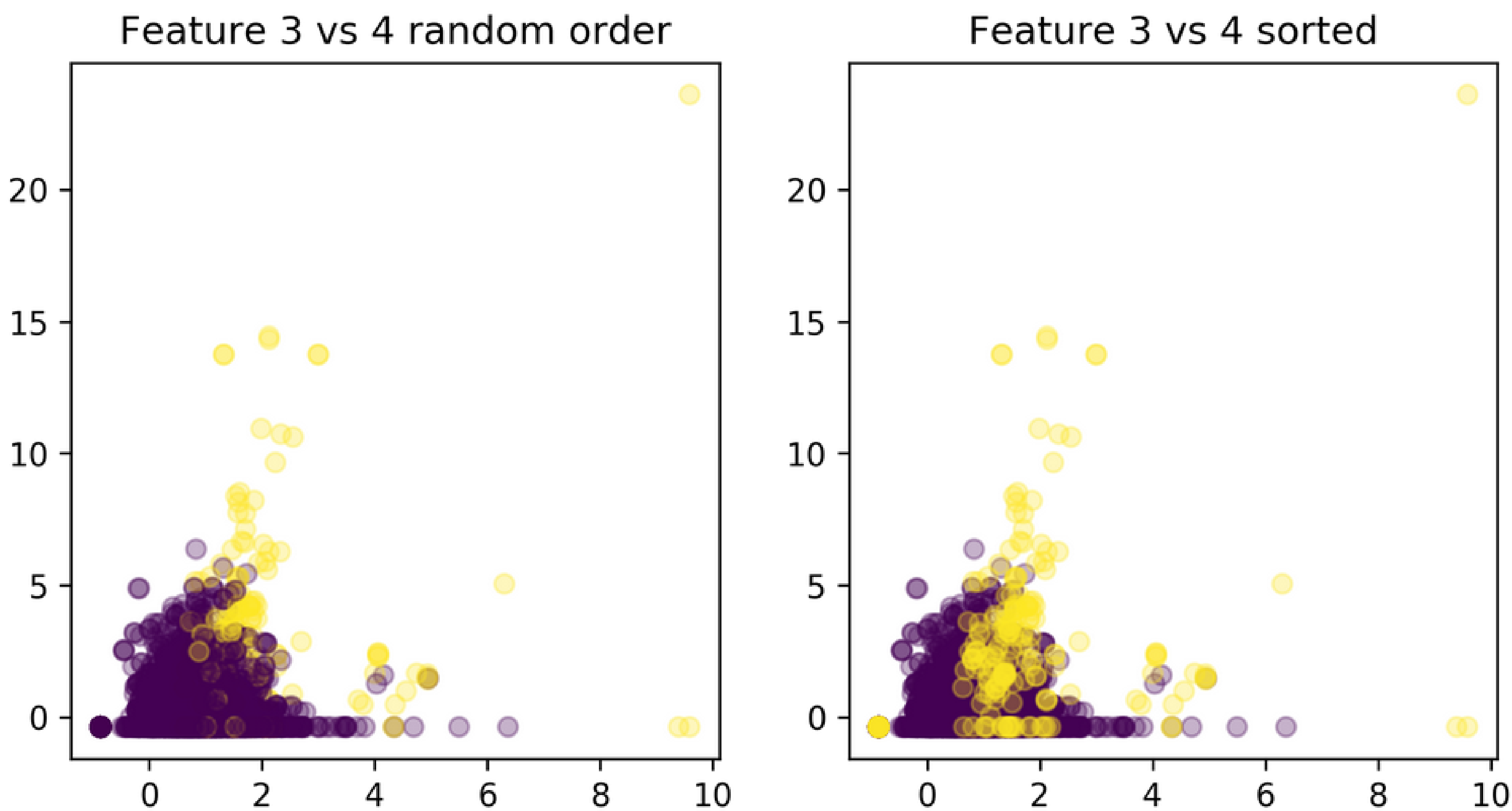

```
from sklearn.linear_model import LogisticRegression
scores = cross_val_score(LogisticRegression(),
                         X_train, y_train, cv=10, scoring='roc_auc')
print(scores.mean())
```

```
0.919622716696
```

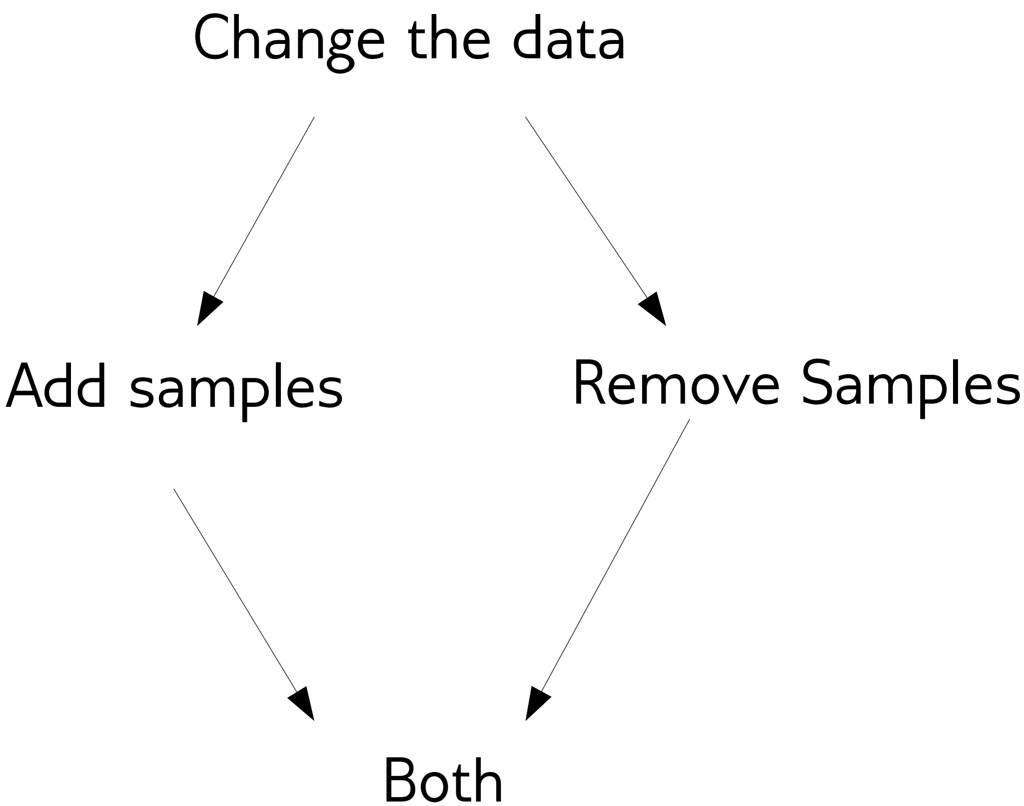
```
from sklearn.ensemble import RandomForestClassifier
scores = cross_val_score(RandomForestClassifier(n_estimators=100),
                         X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.94143939841
```

Mammography data



Basic Approaches



Change the training procedure

Scikit-learn vs resampling

- The transform method only transforms X
- Pipelines work by chaining transforms
- To resample the data, we need to also change y
- Imbalance-learn extends scikit-learn interface with a "sample" method.
- Imbalance-learn has a custom pipeline that allows resampling.
- Imbalance-learn: resampling is only performed during fitting
- Warning: not everything in imbalance-learn is multiclass!

Random Undersampling

- Drop data from the majority class randomly
- Often until balanced
- Very fast training (data shrinks to 2x minority)
- Loses data!

```
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(replacement=False)
X_train_subsample, y_train_subsample = rus.fit_sample(X_train, y_train)
print(X_train.shape)
print(X_train_subsample.shape)
print(np.bincount(y_train_subsample))
```

```
(8387, 6)
(390, 6)
[195 195]
```

Random Undersampling

```python
from imblearn.pipeline import make_pipeline as make_imb_pipeline

undersample_pipe = make_imb_pipeline(RandomUnderSampler(), LogisticRegressionCV())
scores = cross_val_score(undersample_pipe, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.916512922589
```

```python
undersample_pipe = make_imb_pipeline(RandomUnderSampler(), RandomForestClassifier())
scores = cross_val_score(undersample_pipe, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.944496565836
```

As accurate with fraction of samples!
Really good for large datasets!

Random Oversampling

- Repeat samples from the minority class randomly.
- Often until balanced.
- Much slower (dataset grows to 2x majority)

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler()
X_train_oversample, y_train_oversample = ros.fit_sample(X_train, y_train)
print(X_train.shape)
print(X_train_oversample.shape)
print(np.bincount(y_train_oversample))
```

```
(8387, 6)
(16384, 6)
[8192 8192]
```

Random Oversampling

```
oversample_pipe = make_imb_pipeline(RandomOverSampler(), LogisticRegression())
scores = cross_val_score(oversample_pipe, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

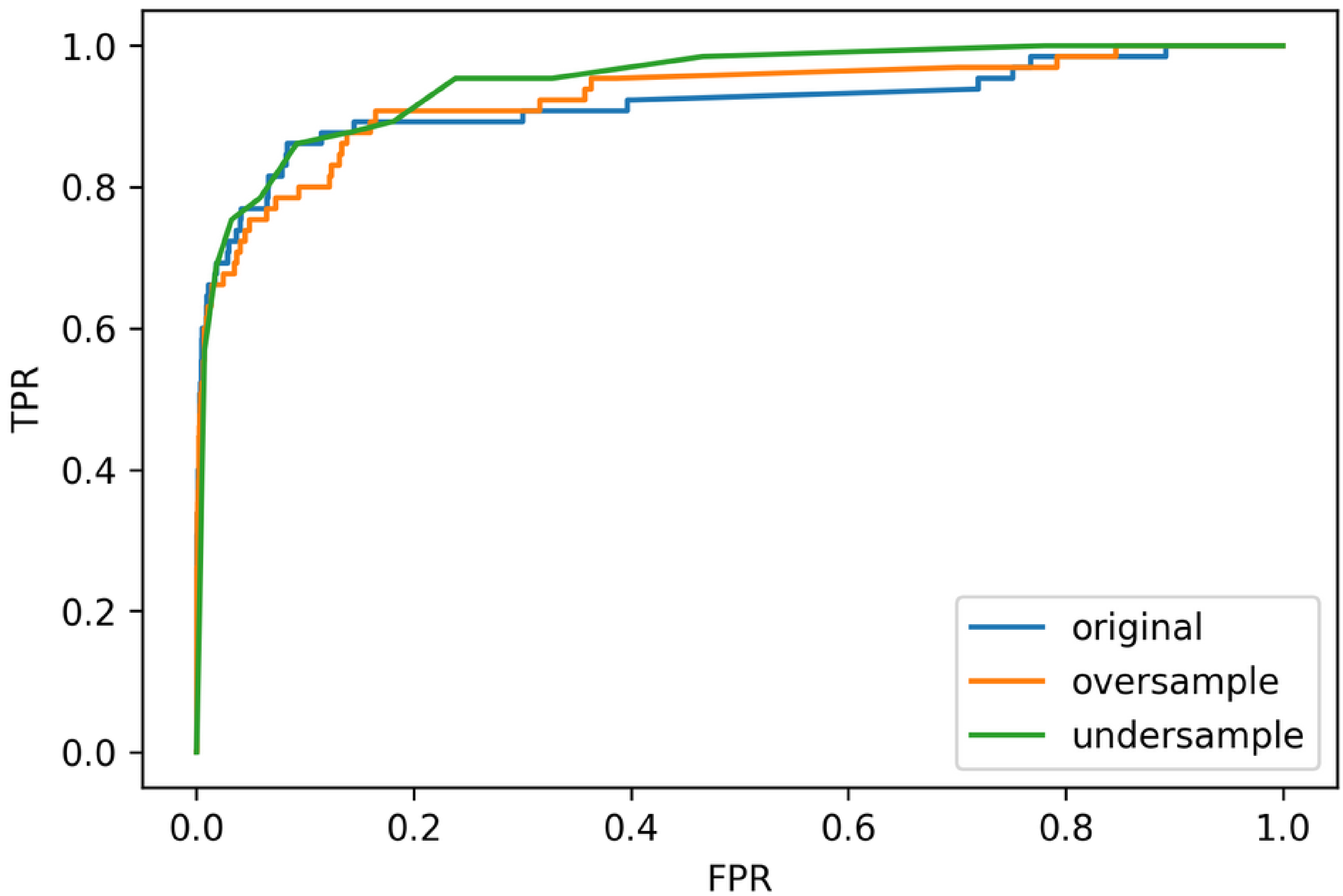
```
0.917755942193
```

```
oversample_pipe_rf = make_imb_pipeline(RandomOverSampler(), RandomForestClassifier())
scores = cross_val_score(oversample_pipe_rf, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

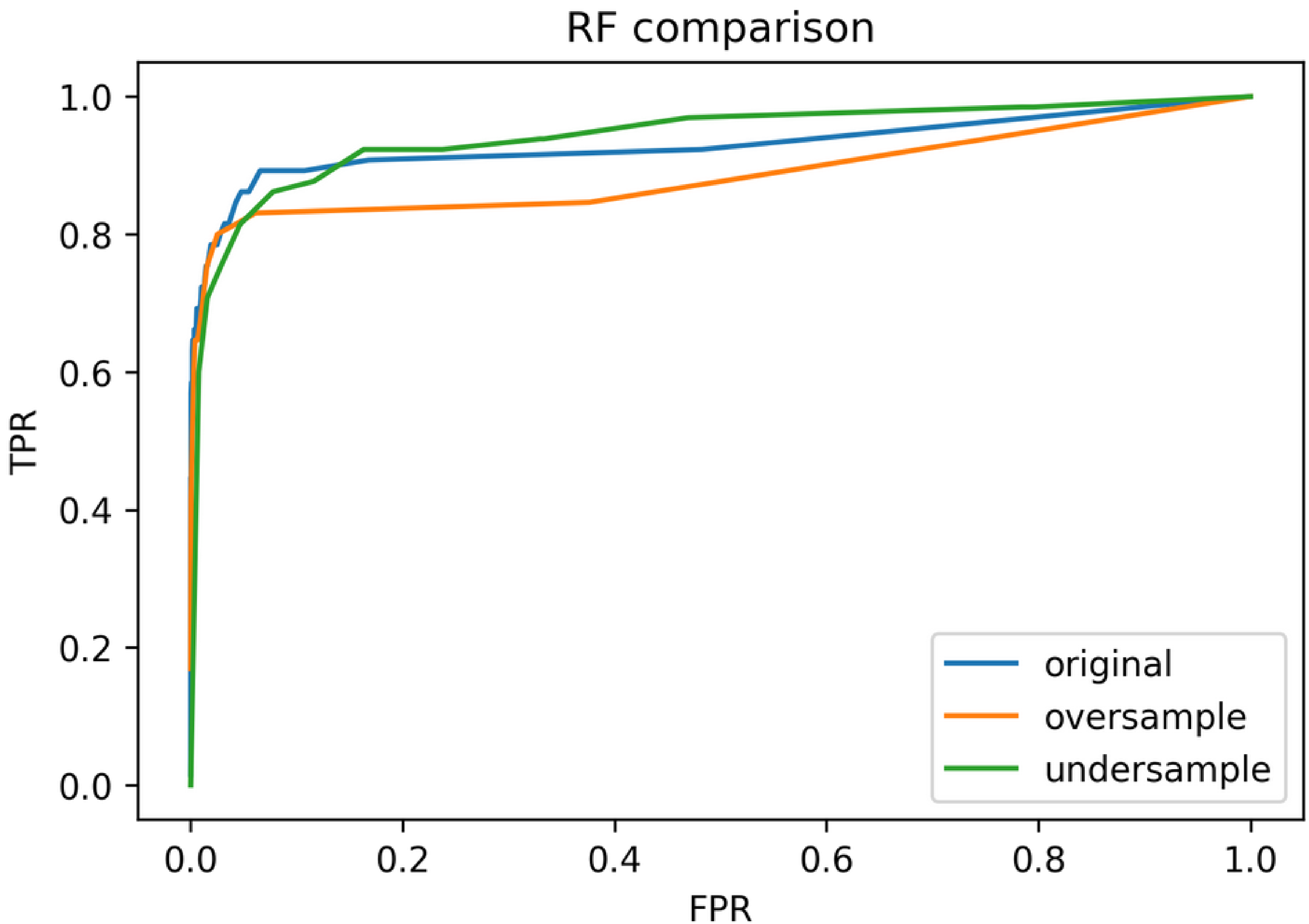
```
0.916313332777
```

Logreg the same, Random Forest much worse than before

ROC Curves for LogReg



ROC Curves for Random Forest



Class-weights

- Instead of repeating samples, re-weight the loss function.
- Works for most models!
- Same effect as over-sampling (though not random), but not as expensive (dataset size the same).

Class-weights in linear models

$$\min_{w \in \mathbb{R}^p} -C \sum_{i=1}^{n} \log(\exp(-y_i w^T \mathbf{x}_i) + 1) + ||w||_2^2$$

$$\min_{w \in \mathbb{R}^p} -\sum_{i=1}^{n} C_{y_i} \log(\exp(-y_i w^T \mathbf{x}_i) + 1) + ||w||_2^2$$

Similar for linear and non-linear SVM

Class weights in trees

Using Class-Weights

```
from sklearn.linear_model import LogisticRegression
scores = cross_val_score(LogisticRegression(class_weight='balanced'),
                         X_train, y_train, cv=10, scoring='roc_auc')
print(scores.mean())
```

```
0.917567920152
```

```
from sklearn.ensemble import RandomForestClassifier
scores = cross_val_score(RandomForestClassifier(n_estimators=100, class_weight='balanced'),
                         X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.91679851501
```

Ensemble Resampling

- Random resampling separate for each instance in an ensemble!
- Paper: “Exploratory Undersampling for Class-Imbalance Learning”
- Not in sklearn (yet), not totally easy with imbalance-learn (but soon).

Quick & Dirty Easy Ensemble

```python
from sklearn.base import clone

def make_resampled_ensemble(estimator, n_estimators=100):
    estimators = []
    for i in range(n_estimators):
        est = clone(estimator)
        if hasattr(est, "random_state"):
            est.random_state = i
        pipe = make_imb_pipeline(RandomUnderSampler(random_state=i, replacement=True),
                                 est)
        estimators.append(("est_i".format(i), pipe))
    return VotingClassifier(estimators, voting="soft")
```

```python
resampled_tree_test = make_resampled_ensemble(DecisionTreeClassifier(max_features='auto'))

scores = cross_val_score(resampled_tree_test, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

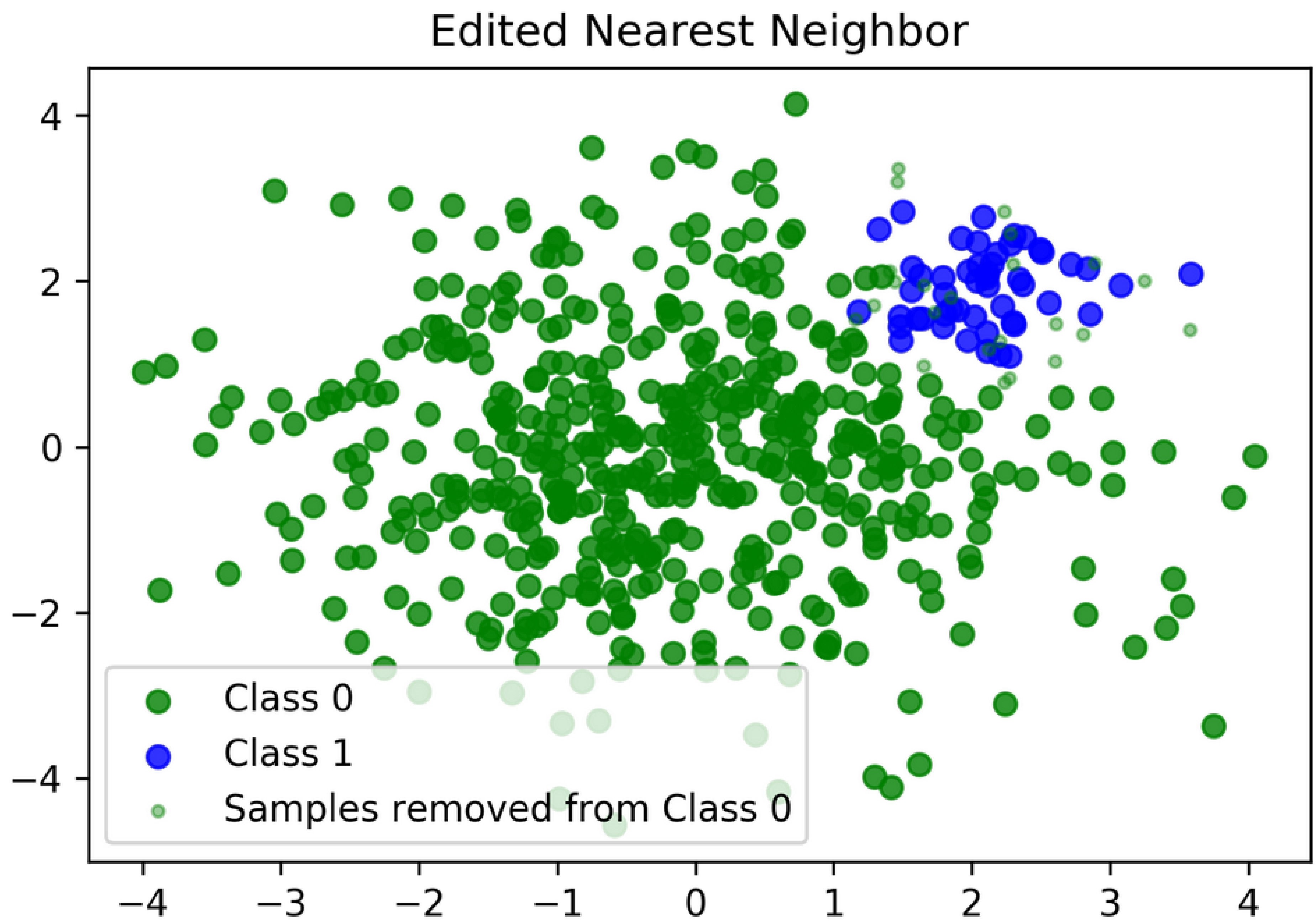
```
0.960342658946
```

As cheap as undersampling, but much better results than anything else!
Didn't do anything for Logistic Regression.

Smart resampling
(based on nearest neighbor heuristics from the 70's

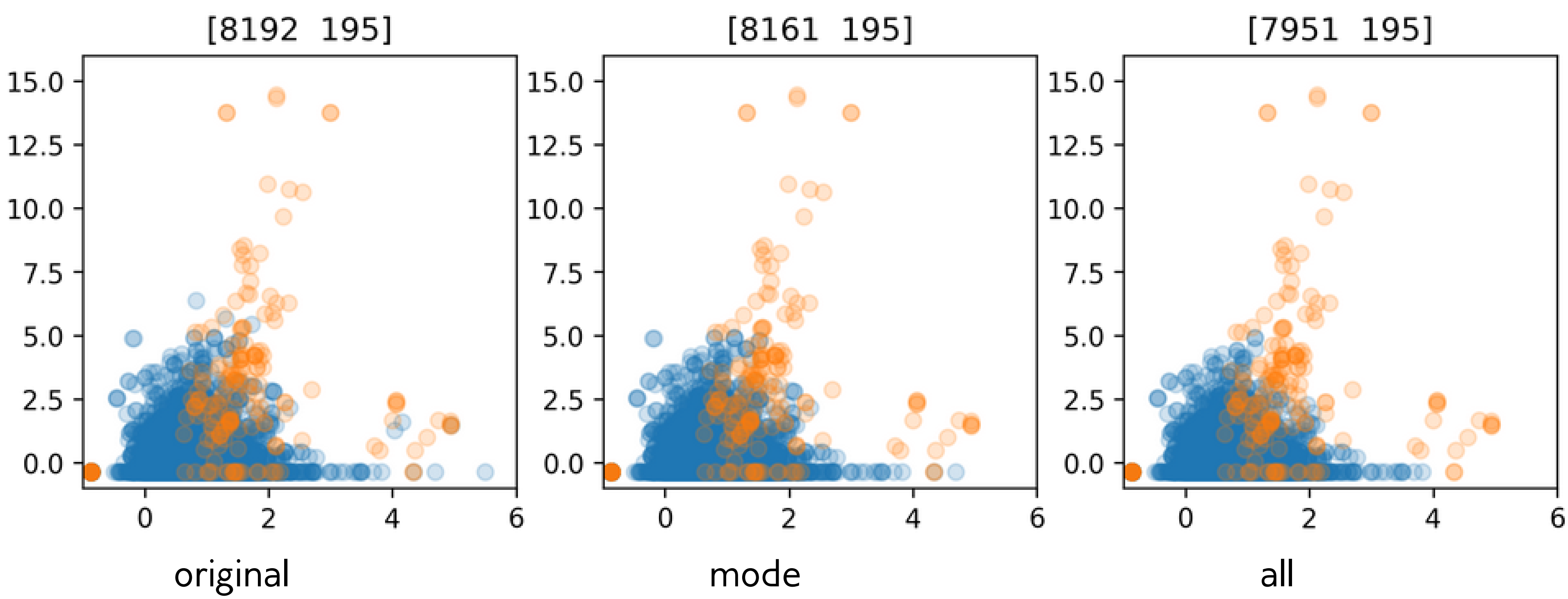
Edited Nearest Neighbors

- Originally as heuristic for reducing dataset for KNN
- Remove all samples that are misclassified by KNN from training data (mode) or that have any point from other class as neighbor (all).
- “Cleans up” outliers and boundaries.


Edited Nearest Neighbor
4
2
0
−2
−4
−4
−3
−2
−1
0
1
2
3
4
Class 0
Class 1
Samples removed from Class 0

Edited Nearest Neighbors

```python
from imblearn.under_sampling import EditedNearestNeighbours
enn = EditedNearestNeighbours(n_neighbors=5)
X_train_enn, y_train_enn = enn.fit_sample(X_train, y_train)

enn_mode = EditedNearestNeighbours(kind_sel="mode", n_neighbors=5)
X_train_enn_mode, y_train_enn_mode = enn_mode.fit_sample(X_train, y_train)
```



```python
enn_pipe = make_imb_pipeline(EditedNearestNeighbours(n_neighbors=5),
                             LogisticRegression())
scores = cross_val_score(enn_pipe, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.920155354576
```

```python
enn_pipe_rf = make_imb_pipeline(EditedNearestNeighbours(n_neighbors=5),
                                RandomForestClassifier(n_estimators=100))
scores = cross_val_score(enn_pipe_rf, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

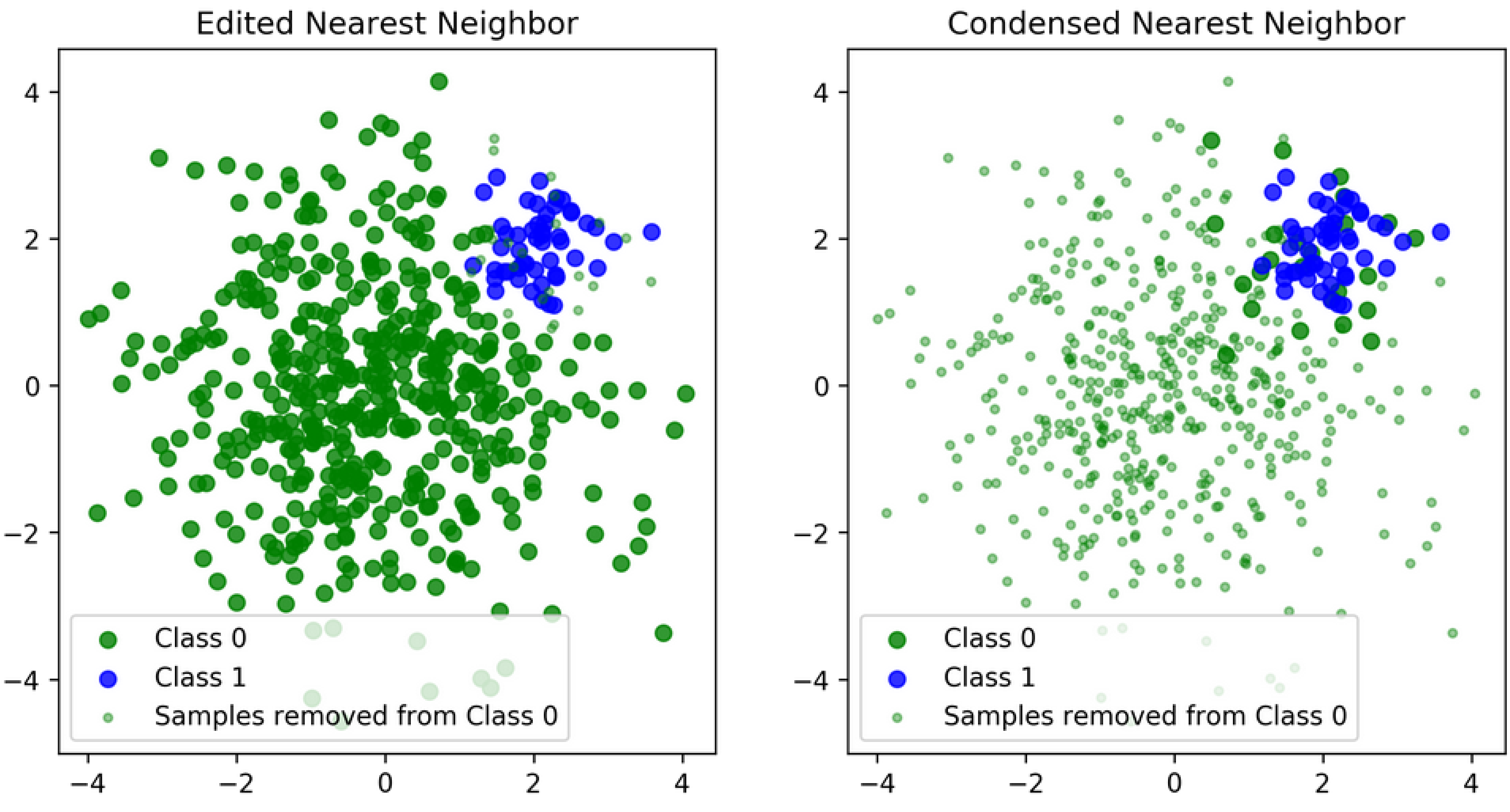
```
0.944075344514
```

Condensed Nearest Neighbors

- Iteratively adds points to the data that are misclassified by KNN
- Focuses on the boundaries
- Usually removes many

```
cnn = CondensedNearestNeighbour()
X_train_cnn, y_train_cnn = cnn.fit_sample(X_train, y_train)
print(X_train_cnn.shape)
print(np.bincount(y_train_cnn))
```

```
(556, 6)
[361 195]
```



Edited Nearest Neighbor
Condensed Nearest Neighbor
4
2
0
−2
−4
−4
−2
0
2
4
Class 0
Class 1
Samples removed from Class 0

```python
cnn_pipe = make_imb_pipeline(CondensedNearestNeighbour(), LogisticRegression())
scores = cross_val_score(cnn_pipe, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.919227113476
```

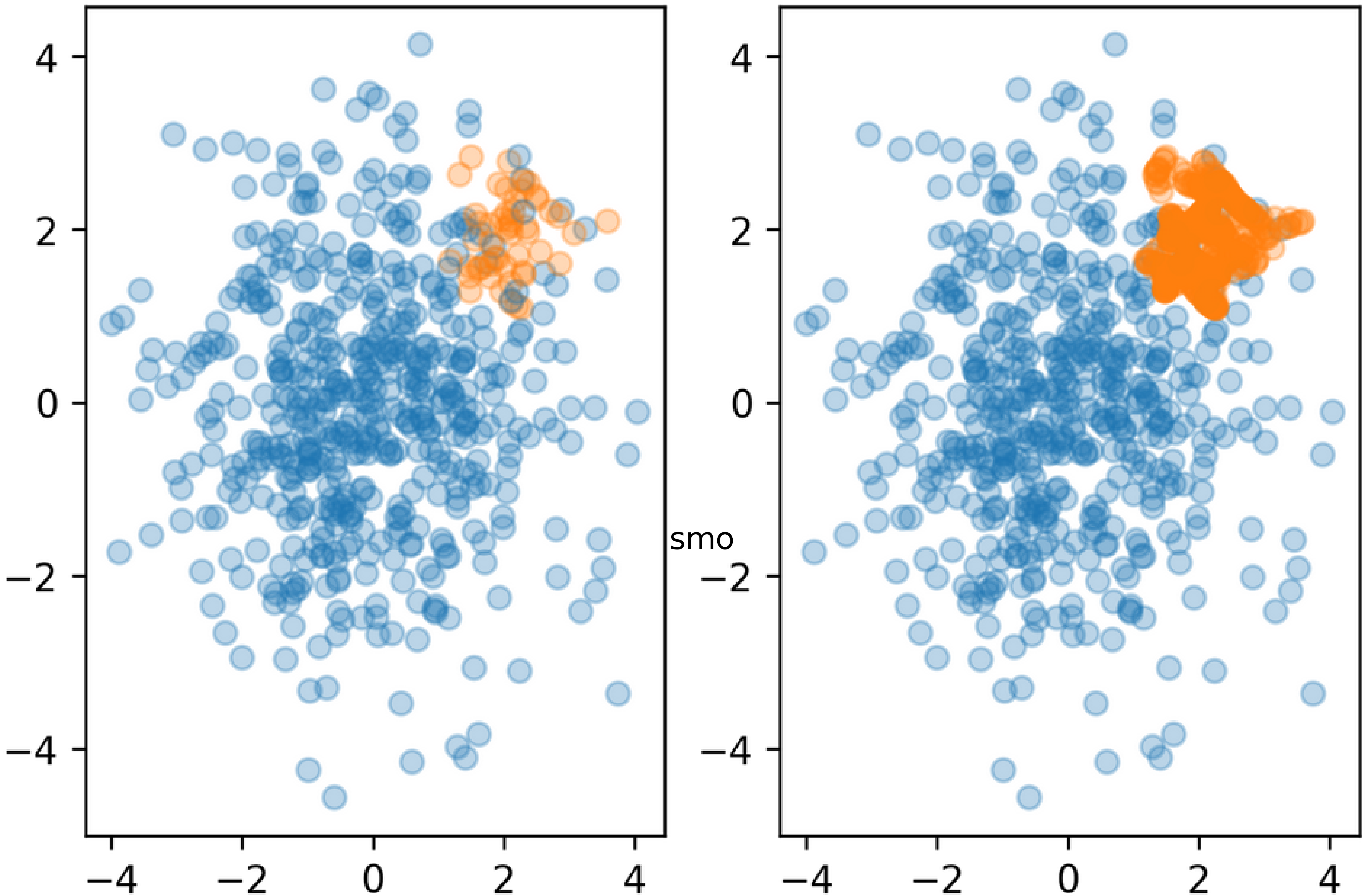
```python
cnn_pipe = make_imb_pipeline(CondensedNearestNeighbour(),
                              RandomForestClassifier(n_estimators=100))
scores = cross_val_score(cnn_pipe, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

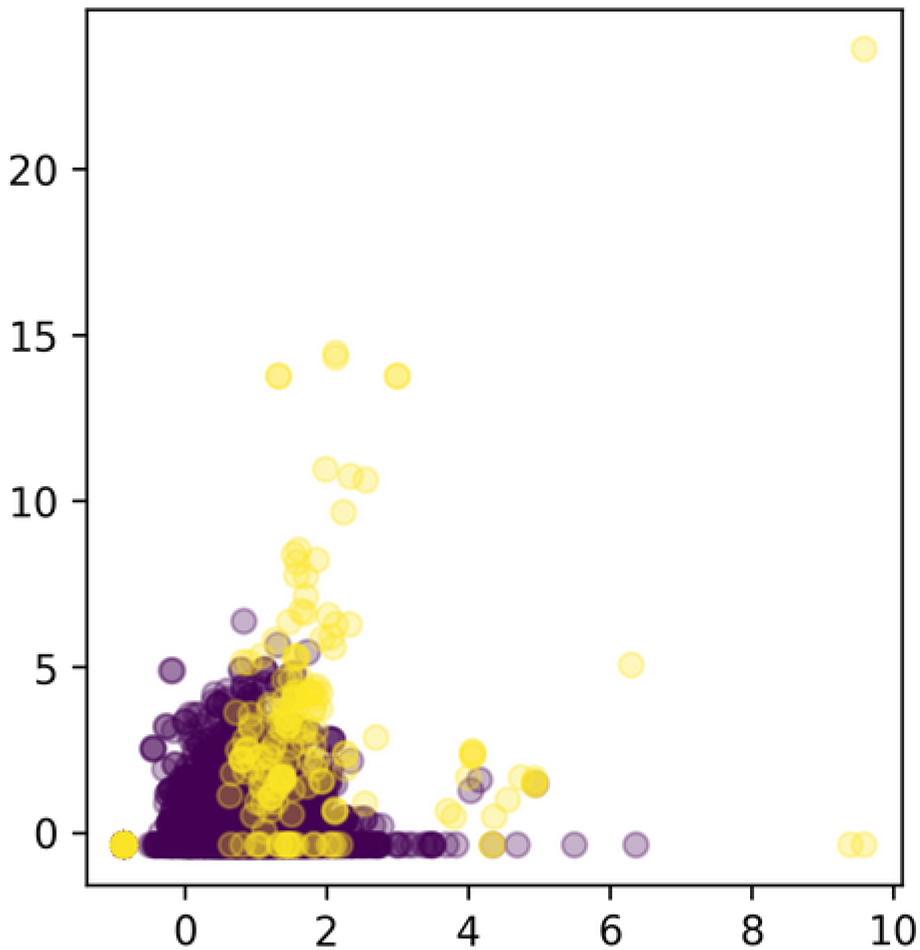
```
0.948040750132
```

Synthetic Sample Generation

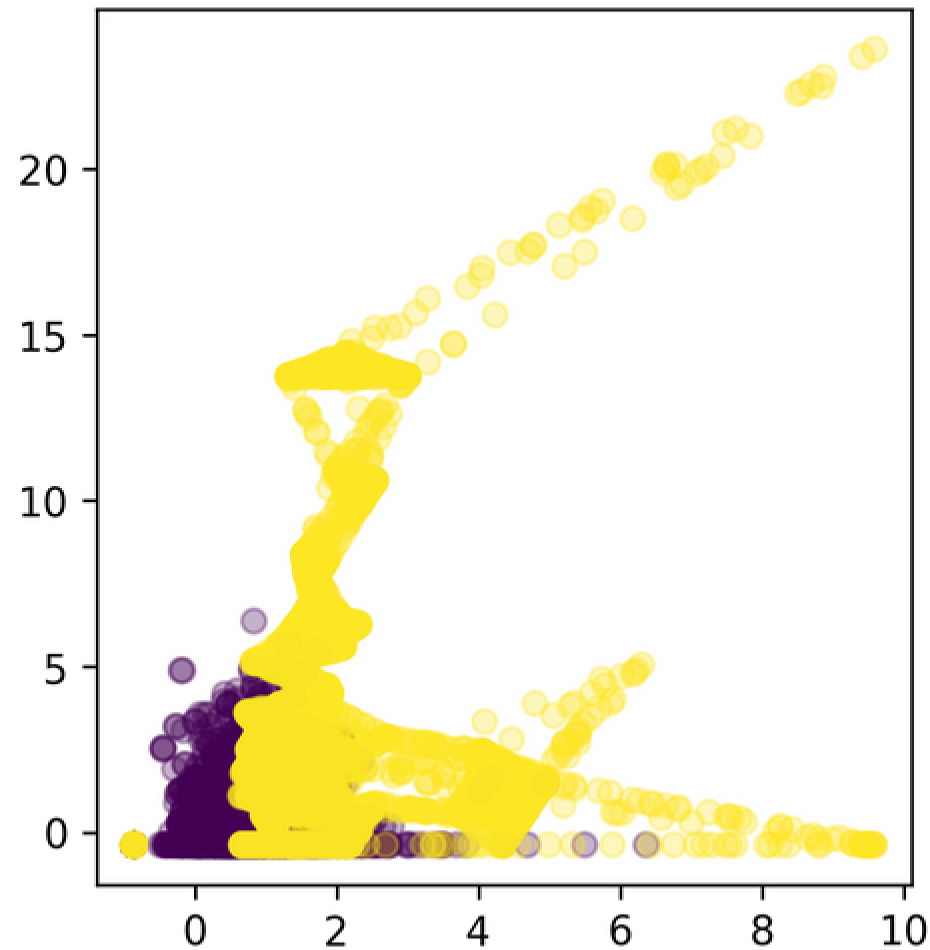
Synthetic Minority Oversampling Technique (SMOTE)

- Adds synthetic interpolated data to smaller class
- For each sample in minority class:
  - Pick random neighbor from k neighbors.
  - Pick point on line connecting the two uniformly
  - Repeat.
- Leads to very large datasets (oversampling)
- Can be combined with undersampling strategies


4
2
0
−2
−4
−4
−2
0
2
4
smo
4
2
0
−2
−4
−4
−2
0
2
4



original



smote

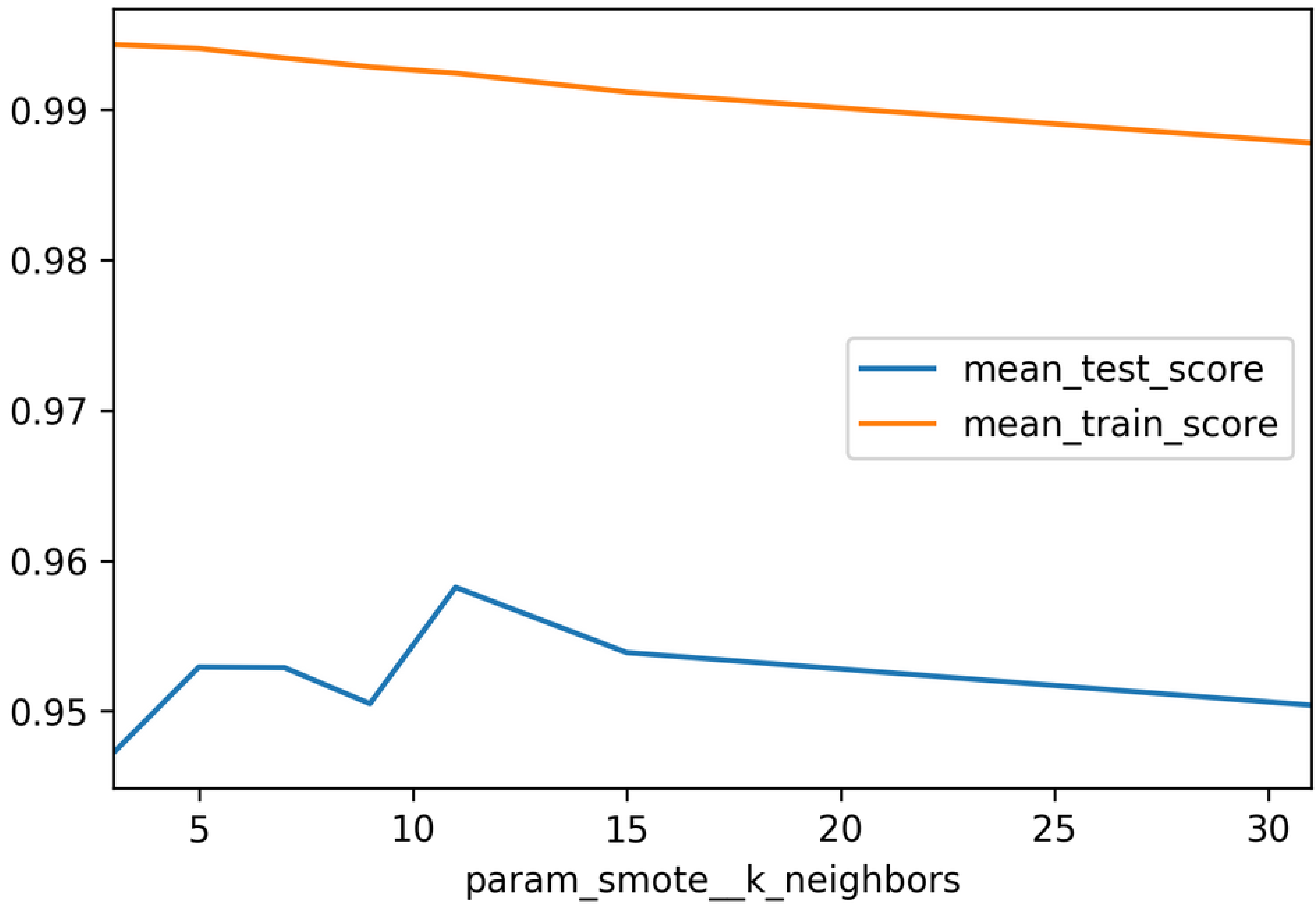
```
smote_pipe = make_imb_pipeline(SMOTE(), LogisticRegression())
scores = cross_val_score(smote_pipe, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.918776908461
```

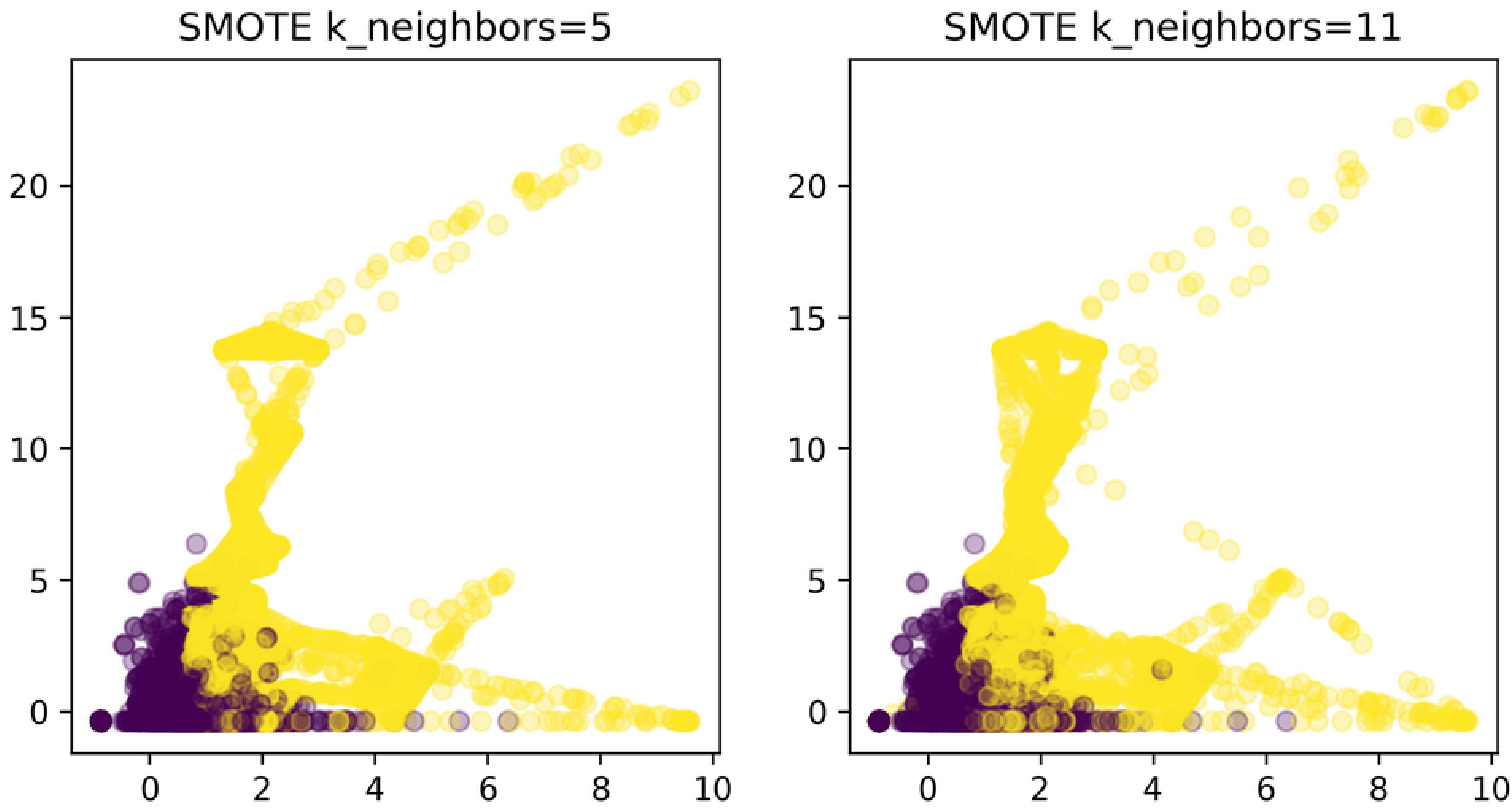
```
smote_pipe_rf = make_imb_pipeline(SMOTE(), RandomForestClassifier(n_estimators=100))
scores = cross_val_score(smote_pipe_rf, X_train, y_train, cv=10, scoring='roc_auc')
print(np.mean(scores))
```

```
0.94679634593
```

```
param_grid = {'smote__k_neighbors': [3, 5, 7, 9, 11, 15, 31]}
search = GridSearchCV(smote_pipe_rf, param_grid, cv=10, scoring="roc_auc")
search.fit(X_train, y_train)
```



0.99
0.98
0.97
0.96
0.95
5
10
15
20
25
30
param_smote__k_neighbors
mean_test_score
mean_train_score


SMOTE k_neighbors=5
SMOTE k_neighbors=11
0
5
10
15
20
0
2
4
6
8
10

Summary

- Always check roc_auc, look at curve
- Undersampling is very fast and can help!
- Undersampling + Ensembles is very powerful!
- Many smart sampling strategies, mixed outcomes
- SMOTE allows adding new interpolated samples, works well in practice
- More advanced variants of SMOTE available