W4995 Applied Machine Learning

Working with Imbalanced Data

03/04/19

Andreas C. Müller

Recap on imbalanced data

Two sources of imbalance

- Asymmetric cost
- Asymmetric data

Why do we care?

- Why should cost be symmetric?
- All data is imbalanced
- Detect rare events

Changing Thresholds

```
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)

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

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

Remedies for the model

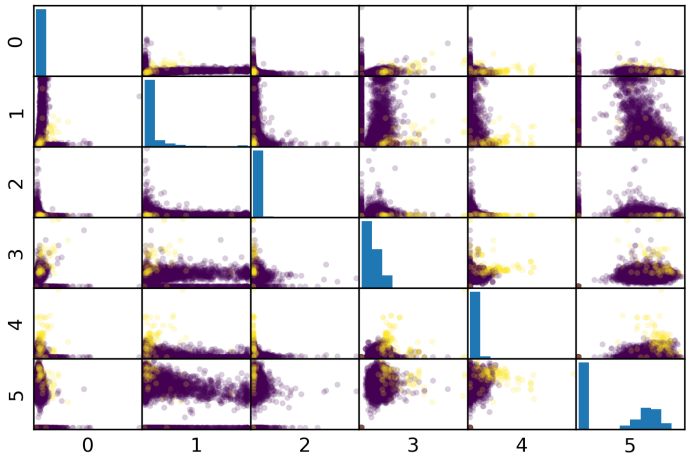
Mammography Data

```
from sklearn.datasets import fetch_openml
# mammography https://www.openml.org/d/310
data = fetch_openml('mammography')
X, y = data.data, data.target
y = (y.astype(np.int) + 1) // 2
X.shape
```

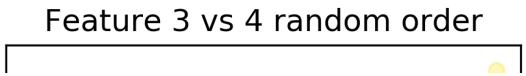
(11183, 6)

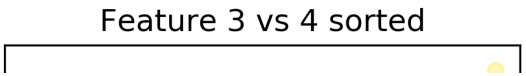
```
np.bincount(y)
```

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



Mammography Data





Mammography Data

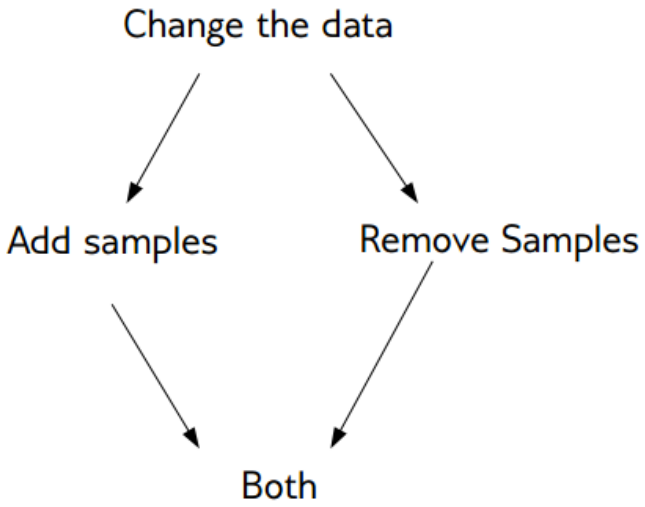
```
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LogisticRegression

scores = cross_validate(LogisticRegression(),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

0.920, 0.630

```
from sklearn.ensemble import RandomForestClassifier
scores = cross_validate(RandomForestClassifier(n_estimators=100),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
```

0.939, 0.722

Basic Approaches



Change the training procedure

Sckit-learn vs resampling

```
pipe = make_pipeline(T1(), T2(), Classifier())
```

Imbalance-Learn

http://imbalanced-learn.org

```
pip install -U imbalanced-learn
```

Extends `sklearn` API

Sampler

To resample a data sets, each sampler implements:

```
data_resampled, targets_resampled = obj.sample(data, targets)
```

Fitting and sampling can also be done in one step:

```
data_resampled, targets_resampled = obj.fit_sample(data, targets)
```

Sampler

To resample a data sets, each sampler implements:

```
data_resampled, targets_resampled = obj.sample(data, targets)
```

Fitting and sampling can also be done in one step:

```
data_resampled, targets_resampled = obj.fit_sample(data, targets)
```

In Pipelines: Sampling only done in `fit`!

Random Undersampling

```
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

```
from imblearn.pipeline import make_pipeline as make_imb_pipeline

undersample_pipe = make_imb_pipeline(RandomUnderSampler(), LogisticRegressionCV())
scores = cross_validate(undersample_pipe,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630
```

```
0.927, 0.527
```

Random Undersampling

```
from imblearn.pipeline import make_pipeline as make_imb_pipeline

undersample_pipe = make_imb_pipeline(RandomUnderSampler(), LogisticRegressionCV())
scores = cross_validate(undersample_pipe,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630
```

```
0.927, 0.527
```

```
undersample_pipe_rf = make_imb_pipeline(RandomUnderSampler(),
                                        RandomForestClassifier())
scores = cross_validate(undersample_pipe_rf,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722
```

```
0.951, 0.629
```

Random Oversampling

```
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_validate(oversample_pipe,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630
```

0.917, 0.585

Random Oversampling

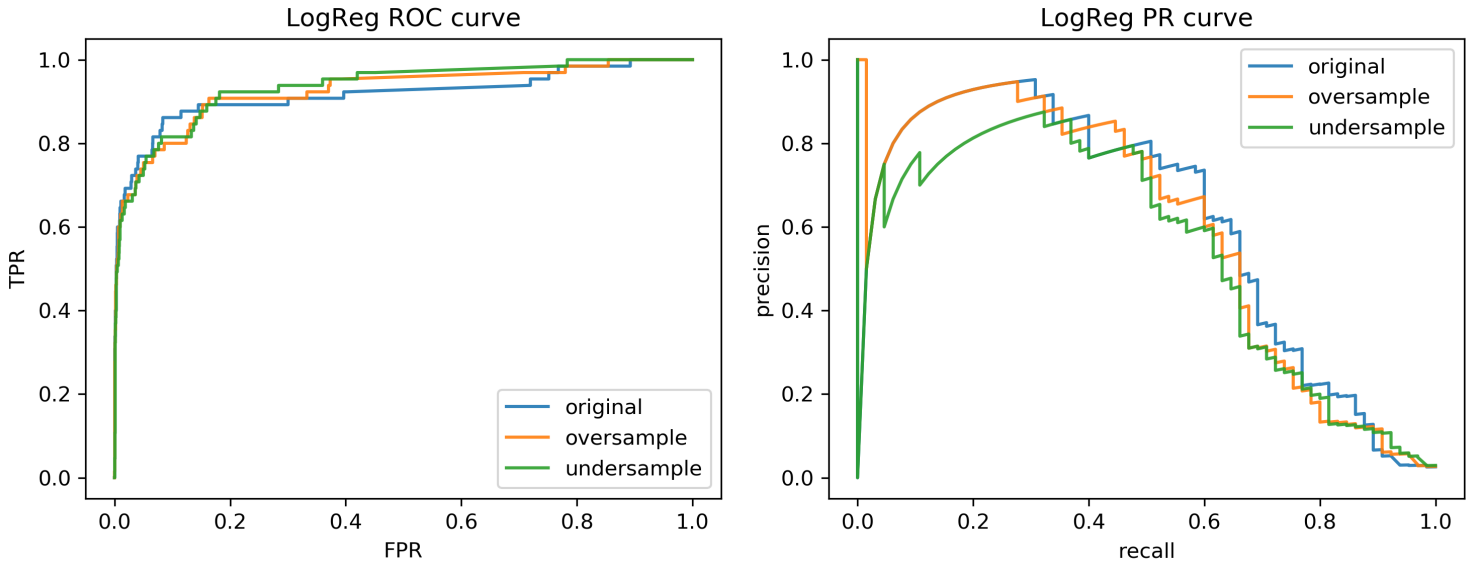
```
oversample_pipe = make_imb_pipeline(RandomOverSampler(), LogisticRegression())
scores = cross_validate(oversample_pipe,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630
```

0.917, 0.585

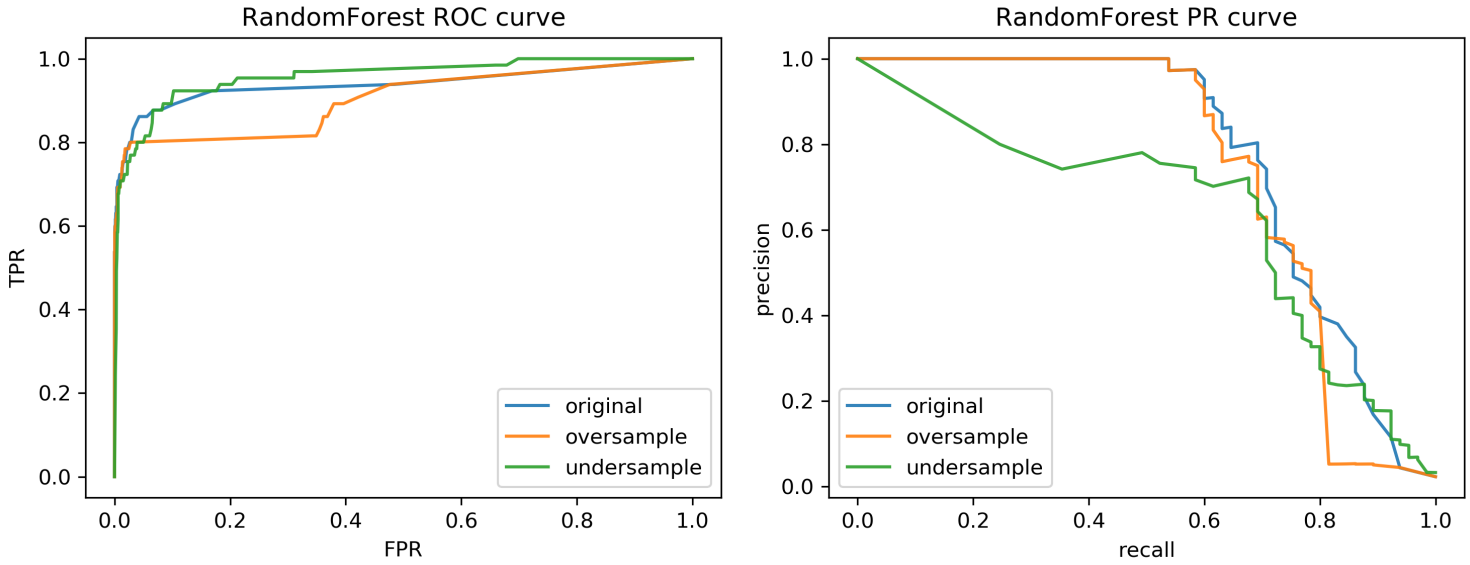
```
oversample_pipe_rf = make_imb_pipeline(RandomOverSampler(),
                                       RandomForestClassifier())
scores = cross_validate(oversample_pipe_rf,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722
```

0.926, 0.715

Curves for LogReg



Curves for Random Forest



ROC or PR?

FPR or Precision?

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

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

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

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

Similar for linear and non-linear SVM

Class weights in trees

Gini Index:

$$H_{\text{gini}}(X_m) = \sum_{k \in \mathcal{Y}} p_{mk}(1 - p_{mk})$$

$$H_{\text{gini}}(X_m) = \sum_{k \in \mathcal{Y}} c_k p_{mk}(1 - p_{mk})$$

Prediction:

Weighted vote

Using Class-Weights

```
scores = cross_validate(LogisticRegression(class_weight='balanced'),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630
```

0.918, 0.587

```
scores = cross_validate(RandomForestClassifier(n_estimators=100,
                                               class_weight='balanced'),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722
```

0.917, 0.701

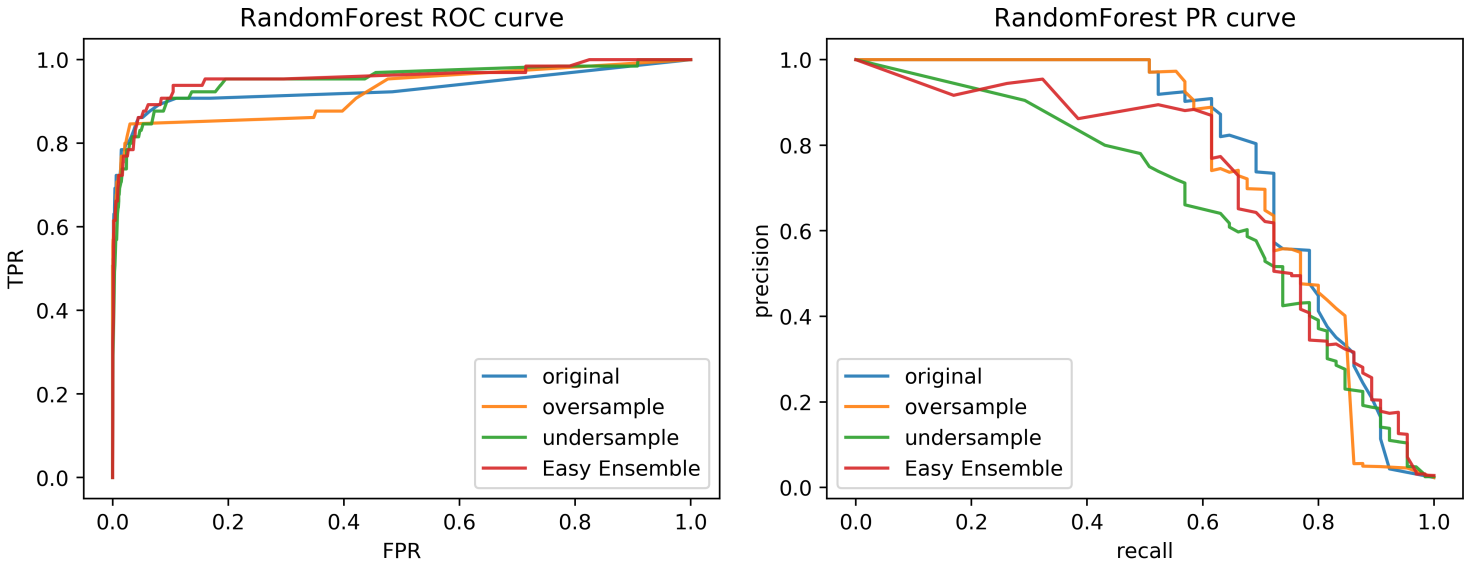
Ensemble Resampling

- Random resampling separate for each instance in an ensemble!
- Chen, Liaw, Breiman: “Using random forest to learn imbalanced data.”
- Paper: “Exploratory Undersampling for Class Imbalance Learning”
- Not in sklearn (yet)
- Easy with imblearn

Easy Ensemble with imblearn

```
from sklearn.tree import DecisionTreeClassifier
from imblearn.ensemble import BalancedBaggingClassifier

# from imblearn.ensemble import BalancedRandomForestClassifier
# resampled_rf = BalancedRandomForestClassifier()

tree = DecisionTreeClassifier(max_features='auto')
resampled_rf = BalancedBaggingClassifier(base_estimator=tree,
                                         n_estimators=100, random_state=0)
scores = cross_validate(resampled_rf,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722
```

0.957, 0.654


RandomForest ROC curve
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
original
oversample
undersample
Easy Ensemble
RandomForest PR curve
precision
recall
0.0
0.2
0.4
0.6
0.8
1.0
original
oversample
undersample
Easy Ensemble

Smart resampling

(based on nearest neighbour heuristics from the 70's)

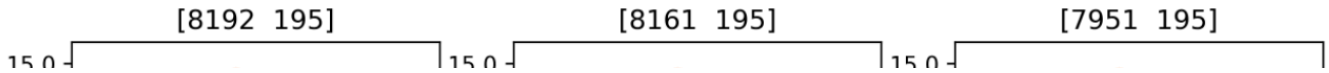
Edited Nearest Neighbours

- 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

Edited Nearest Neighbours

```
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)
```



Edited Nearest Neighbours

```
enn_pipe = make_imb_pipeline(EditedNearestNeighbours(n_neighbors=5),
                             LogisticRegression())
scores = cross_validate(enn_pipe, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.920, 0.630
```

0.920, 0.627

```
enn_pipe_rf = make_imb_pipeline(EditedNearestNeighbours(n_neighbors= 5),
                                RandomForestClassifier(n_estimators=100))
scores = cross_validate(enn_pipe_rf, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()
# baseline was 0.939, 0.722
```

0.940, 0.687

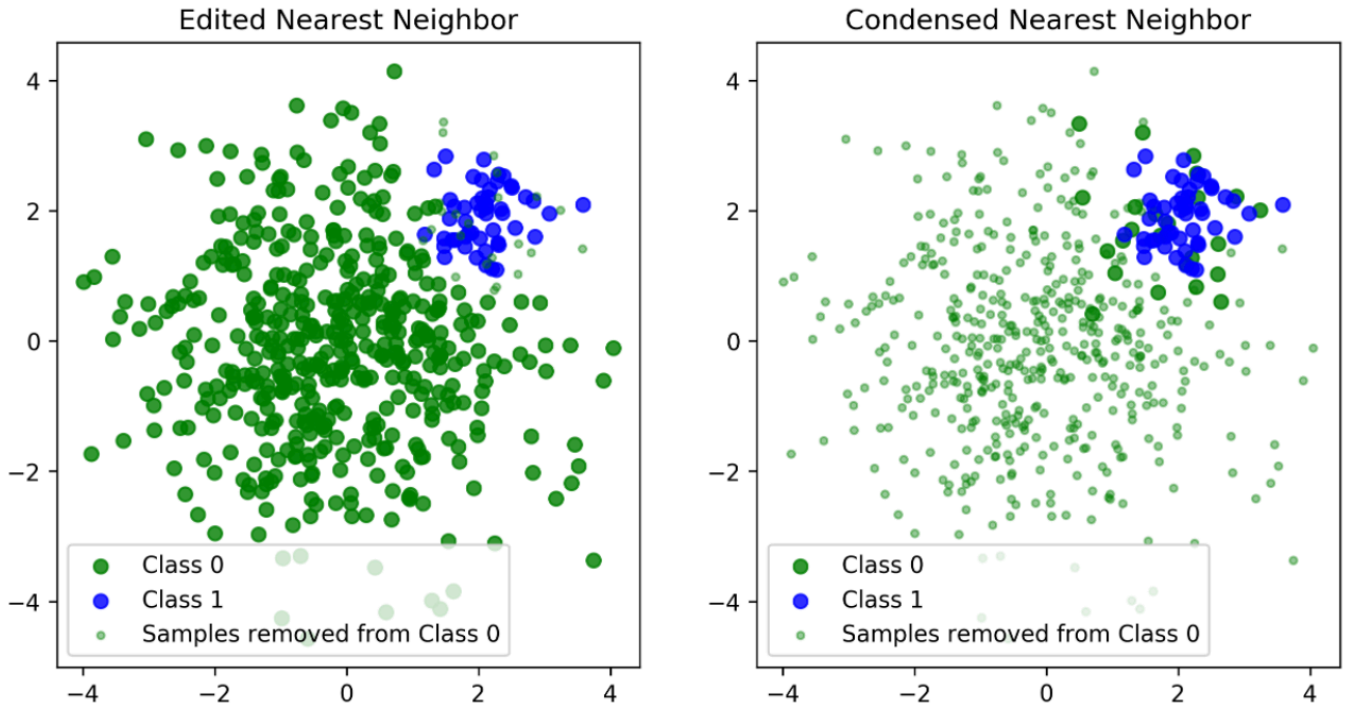
Condensed Nearest Neigbours

- 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
Class 0
Class 1
Samples removed from Class 0
4
2
0
−2
−4
−4
−2
0
2
4

```
cnn_pipe = make_imb_pipeline(CondensedNearestNeighbour(), LogisticRegression())
scores = cross_validate(cnn_pipe, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
pd.DataFrame(scores)[['test_roc_auc', 'test_average_precision']].mean()
# baseline was 0.920, 0.630
```

```
test_roc_auc              0.920
test_average_precision    0.606
```

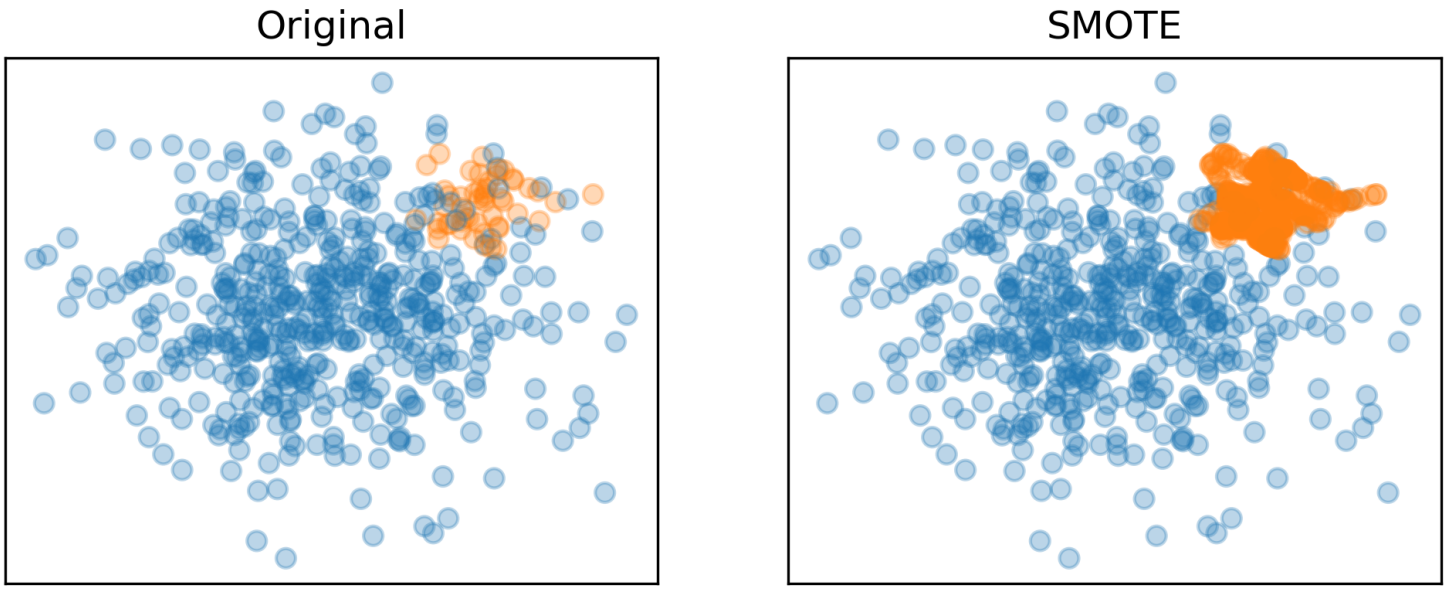
```
cnn_pipe_rf = make_imb_pipeline(CondensedNearestNeighbour(),
                                RandomForestClassifier(n_estimators=100))
scores = cross_validate(cnn_pipe_rf, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
pd.DataFrame(scores)[['test_roc_auc', 'test_average_precision']].mean()
# baseline was 0.939, 0.722
```

```
test_roc_auc              0.953
test_average_precision    0.701
```

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 (or within rectangle)
  – Repeat.


Original
SMOTE


Original
SMOTE

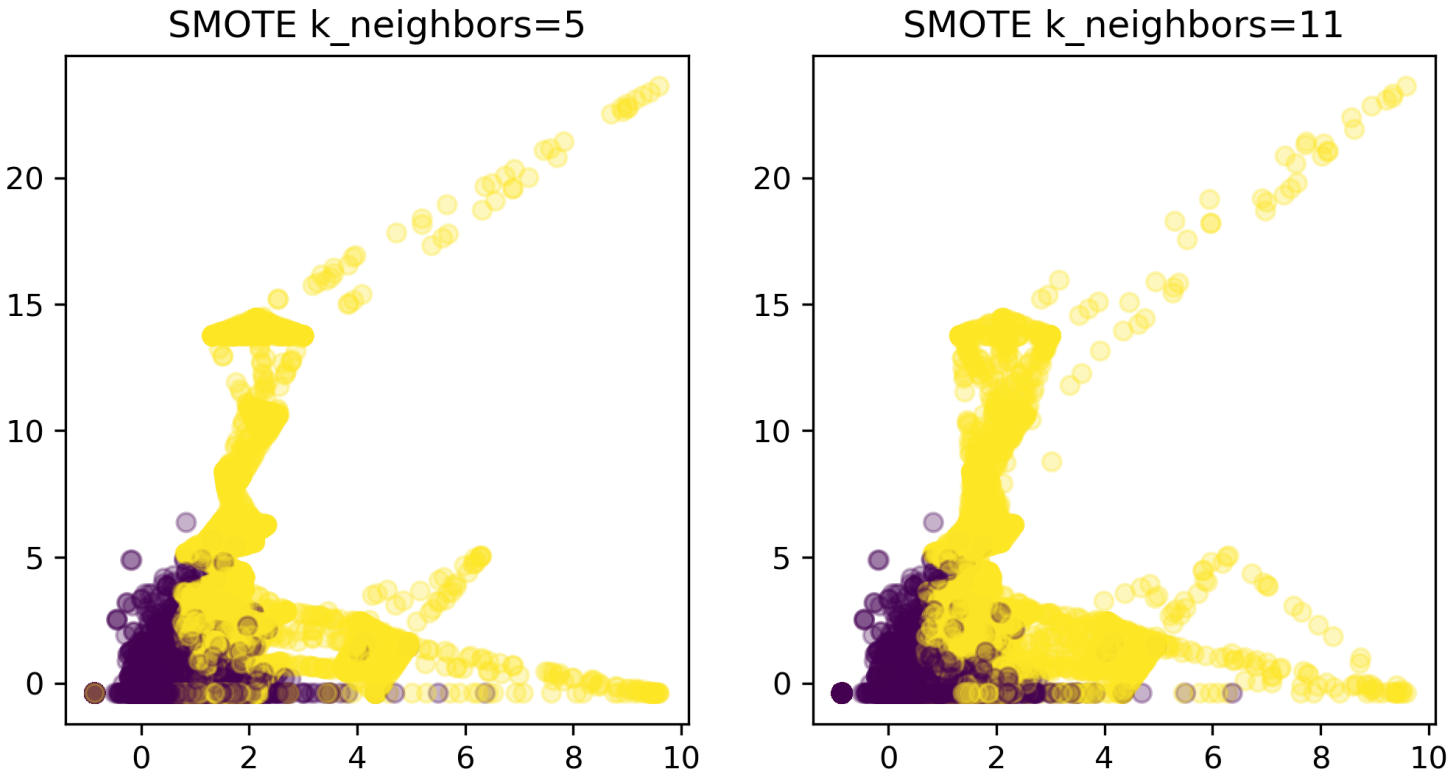
```
smote_pipe = make_imb_pipeline(SMOTE(), LogisticRegression())
scores = cross_validate(smote_pipe, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
pd.DataFrame(scores)[['test_roc_auc', 'test_average_precision']].mean()
# baseline was 0.920, 0.630
```

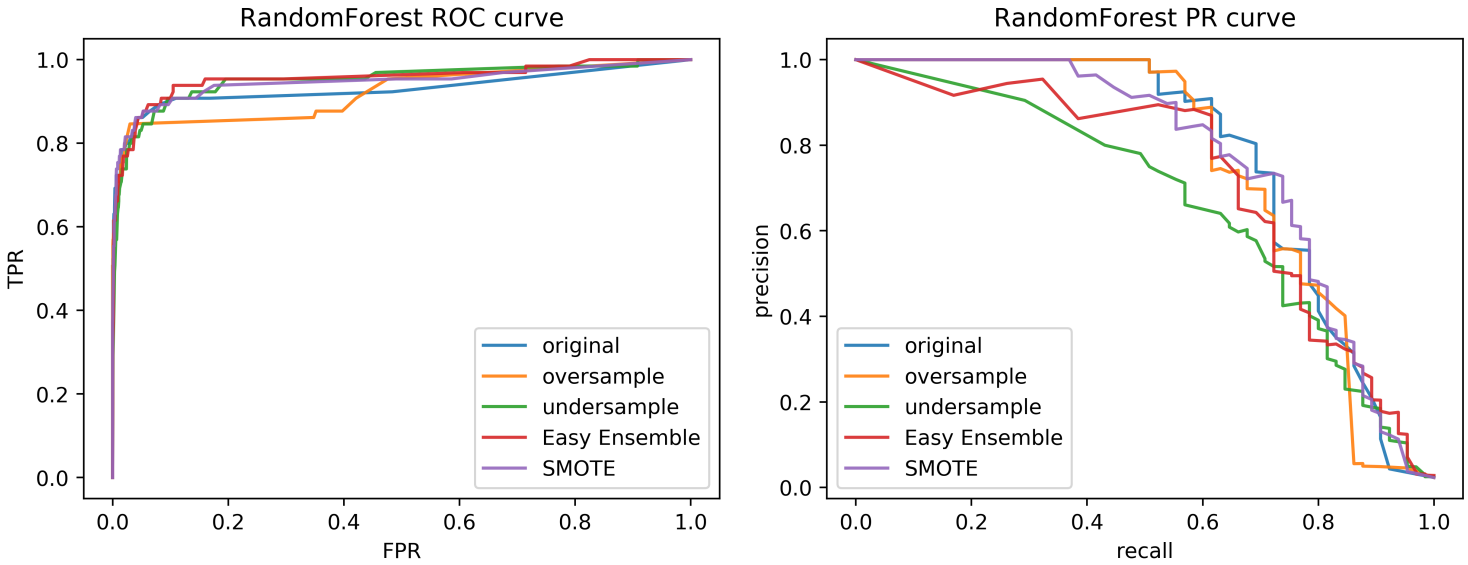
0.919, 0.585

```
smote_pipe_rf = make_imb_pipeline(SMOTE(),
                                  RandomForestClassifier(n_estimators=100))
scores = cross_validate(smote_pipe_rf, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
pd.DataFrame(scores)[['test_roc_auc', 'test_average_precision']].mean()
# baseline was 0.939, 0.722
```

0.946, 0.688

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


RandomForest ROC curve
TPR
FPR
0.0
0.2
0.4
0.6
0.8
1.0
original
oversample
undersample
Easy Ensemble
SMOTE
RandomForest PR curve
precision
recall
0.0
0.2
0.4
0.6
0.8
1.0
original
oversample
undersample
Easy Ensemble
SMOTE

Summary

- Always check roc_auc an AP, look at curves
- Undersampling is very fast and can help!
- Undersampling + Ensembles worth a try.
- Many smart sampling strategies, mixed outcomes
- SMOTE allows adding new interpolated samples
- Mixed outcomes with SMOTE, also definition a bit unclear