RESAMPLING TECHNIQUES AND OTHER STRATEGIES FOR HANDLING UNBALANCED DATASETS IN CLASSIFICATION

Ajinkya More

PyData, San Francisco 2016

INTRODUCTION

- A number of classification problems need to deal with data imbalance amongst classes
- One class may have significantly higher/lower representation in the training data relative to others
- Often performance on the minority class if of higher interest
- Vanilla classification techniques may be inadequate in achieving this goal

He, Haibo, and Edwardo A. Garcia. "Learning from imbalanced data." IEEE Transactions on knowledge and data engineering 21.9 (2009): 1263-1284.

EXAMPLES

- Fraud detection
- Product categorization
- Disease diagnosis

HANDLING DATA IMBALANCE

- Modify the loss function
- Modify the dataset (resampling)
- Ensemble methods

NOTATION AND METRICS

- Majority class: L
- Minority class: S
- Objectives:
  - High recall on S
  - High precision on L

In [2]:

```
from sklearn.datasets import make_classification
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.grid_search import GridSearchCV
from sklearn.cross_validation import KFold, train_test_split
import numpy as np
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler, NearMiss, CondensedNearest
Neighbour
from imblearn.under_sampling import EditedNearestNeighbours, RepeatedEditedNearest
Neighbours, TomekLinks
from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.combine import SMOTEENN, SMOTETomek
from imblearn.ensemble import BalanceCascade, EasyEnsemble
from sklearn.ensemble import AdaBoostClassifier
import warnings
from itable import PrettyTable, TableStyle, CellStyle
import pandas as pd
warnings.filterwarnings('ignore')
%pylab inline
pylab.rcParams['figure.figsize'] = (12, 6)
plt.style.use('fivethirtyeight')
```

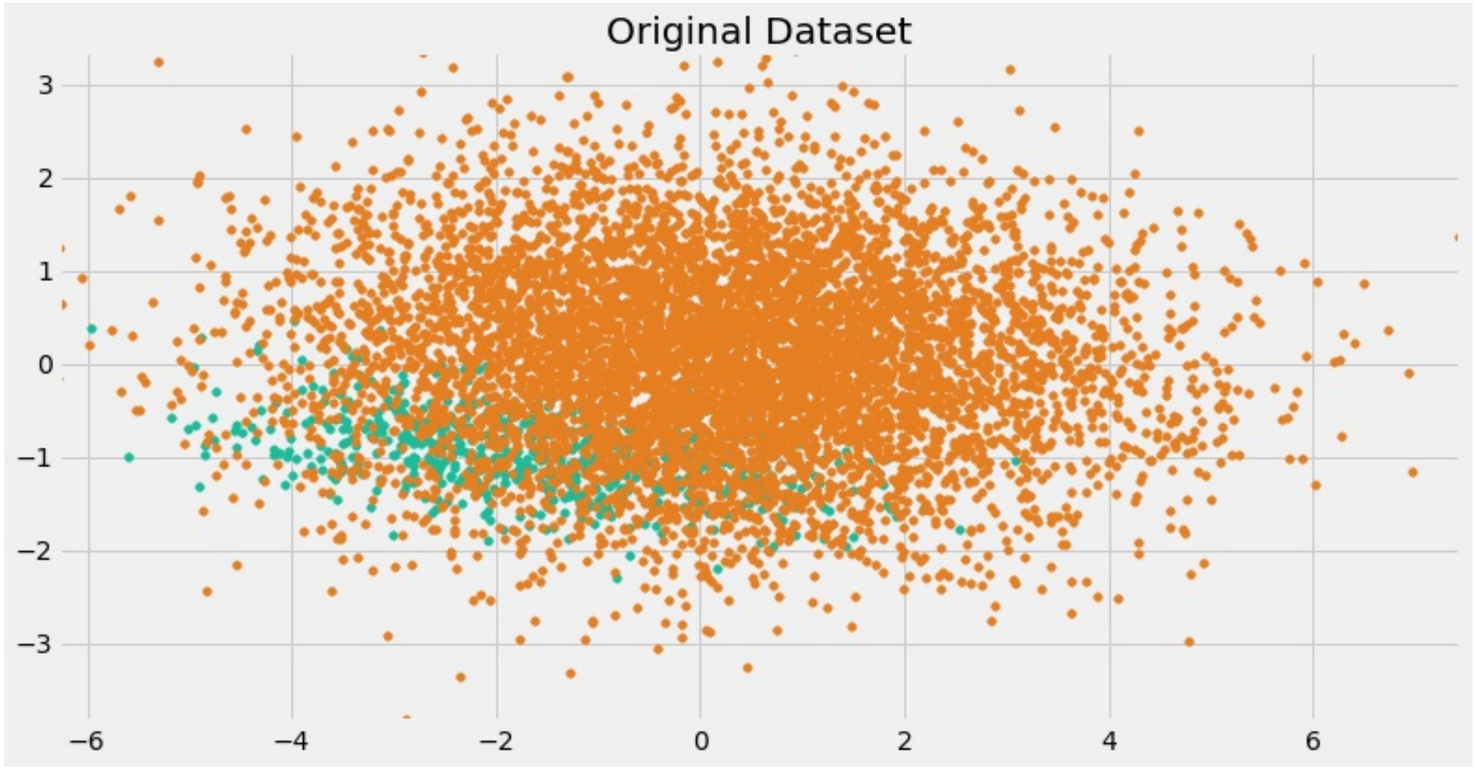
```
Populating the interactive namespace from numpy and matplotlib
```

In [4]:

```
# Generate data with two classes
X, y = make_classification(class_sep=1.2, weights=[0.1, 0.9], n_informative=3,
                           n_redundant=1, n_features=5, n_clusters_per_class=1,
                           n_samples=10000, flip_y=0, random_state=10)
pca = PCA(n_components=2)
X = pca.fit_transform(X)

y = y.astype('str')
y[y=='1'] ='L'
y[y=='0'] ='S'

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
                                                    random_state=0)

X_1, X_2 = X_train[y_train=='S'], X_train[y_train=='L']
```

In [5]:

```
# Scatter plot of the data
plt.scatter(zip(*X_1)[0], zip(*X_1)[1], color='#1abc9c')
plt.scatter(zip(*X_2)[0], zip(*X_2)[1], color='#e67e22')

x_coords = zip(*X_1)[0] + zip(*X_2)[0]
y_coords = zip(*X_1)[1] + zip(*X_2)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Original Dataset")
plt.show()
```



In [6]:

```
# Fit a Logistic Regression model
clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train, y_train)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]
```

In [7]:

```
plt.scatter(zip(*X_1)[0], zip(*X_1)[1], color='#1abc9c')
plt.scatter(zip(*X_2)[0], zip(*X_2)[1], color='#e67e22')

x_coords = zip(*X_1)[0] + zip(*X_2)[0]
y_coords = zip(*X_1)[1] + zip(*X_2)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Original Dataset - Fitted Logistic Regression")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```

In [8]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.90      0.98      0.94      2680
          S       0.39      0.12      0.19       320

avg / total       0.85      0.89      0.86      3000
```

ASYMMETRIC LOSS FUNCTIONS

- Penalize misclassifications on S higher
- Logistic loss

$$-\Sigma_{j \in \mathcal{C}} \Sigma_{y_i = j} ln(P(y_i = j | x_i; \theta))$$

- Weighted logistic loss

$$-\Sigma_{j \in \mathcal{C}} \Sigma_{y_i = j} w_j ln(P(y_i = j | x_i; \theta))$$

King, Gary, and Langche Zeng. "Logistic regression in rare events data." Political analysis 9.2 (2001): 137-163.

In [10]:

```
# Logistic Regression with balanced class weights
clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2'],
        'class_weight': ['balanced']}

cv = KFold(X_train.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train, y_train)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_
```

In [11]:

```
x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

plt.scatter(zip(*X_1)[0], zip(*X_1)[1], color='#1abc9c')
plt.scatter(zip(*X_2)[0], zip(*X_2)[1], color='#e67e22')

x_coords = zip(*X_1)[0] + zip(*X_2)[0]
y_coords = zip(*X_1)[1] + zip(*X_2)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Original Dataset - Fitted Logistic Regression")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```

In [13]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.98      0.79      0.88      2680
          S       0.34      0.89      0.49       320

avg / total       0.92      0.80      0.84      3000
```

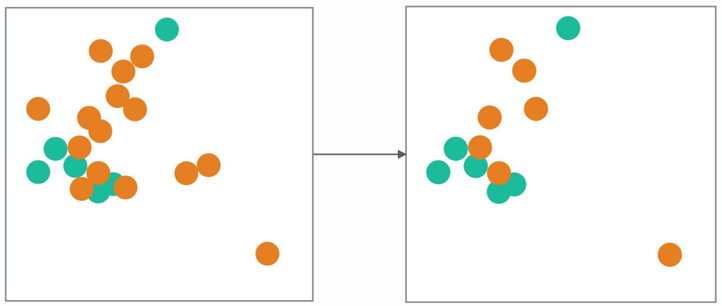
In [14]:
```
precision_recall_comparison
```

Out[14]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12

RANDOM UNDERSAMPLING

- Randomly undersample examples from L
- May lose informative examples

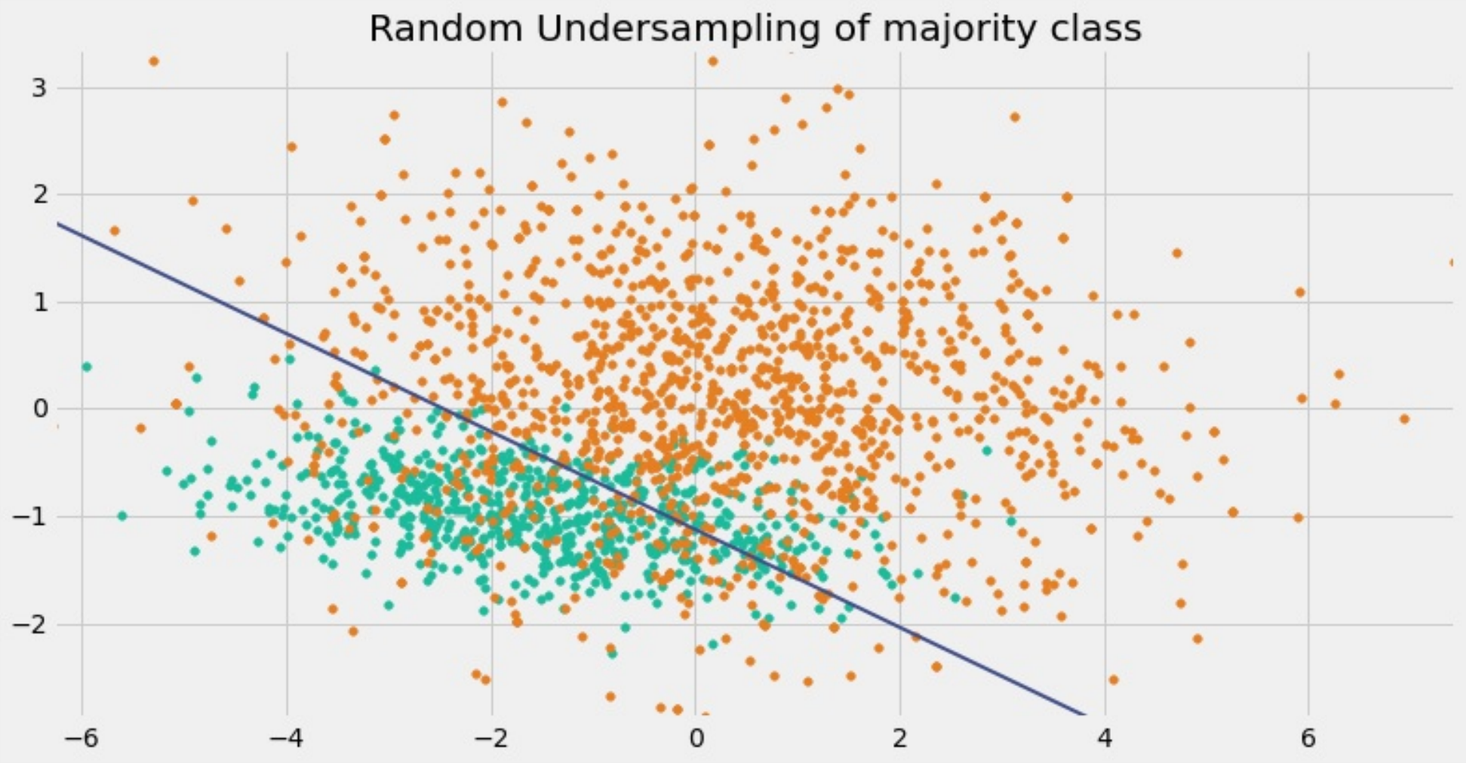


In [15]:
```
# Random undersampling of majority class
us = RandomUnderSampler(ratio=0.5, random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 1360, 'S': 680})
```

In [16]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Random Undersampling of majority class")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [18]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.97      0.83      0.90      2680
          S       0.36      0.82      0.50       320

avg / total       0.91      0.83      0.85      3000
```

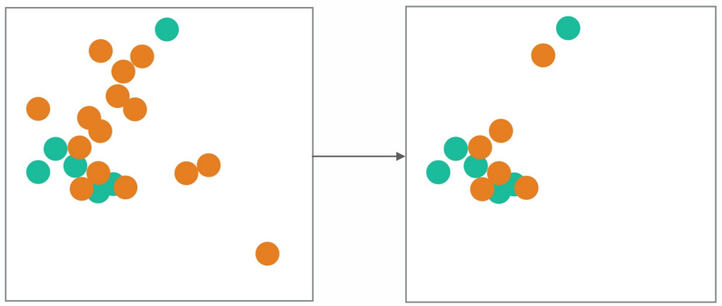
In [19]:
```
precision_recall_comparison
```

Out[19]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89

NEARMISS-1

- Retain points from L whose mean distance to the k nearest points in S is lowest



Mani, Inderjeet, and I. Zhang. "kNN approach to unbalanced data distributions: a case study involving information extraction." Proceedings of workshop on learning from imbalanced datasets. 2003.

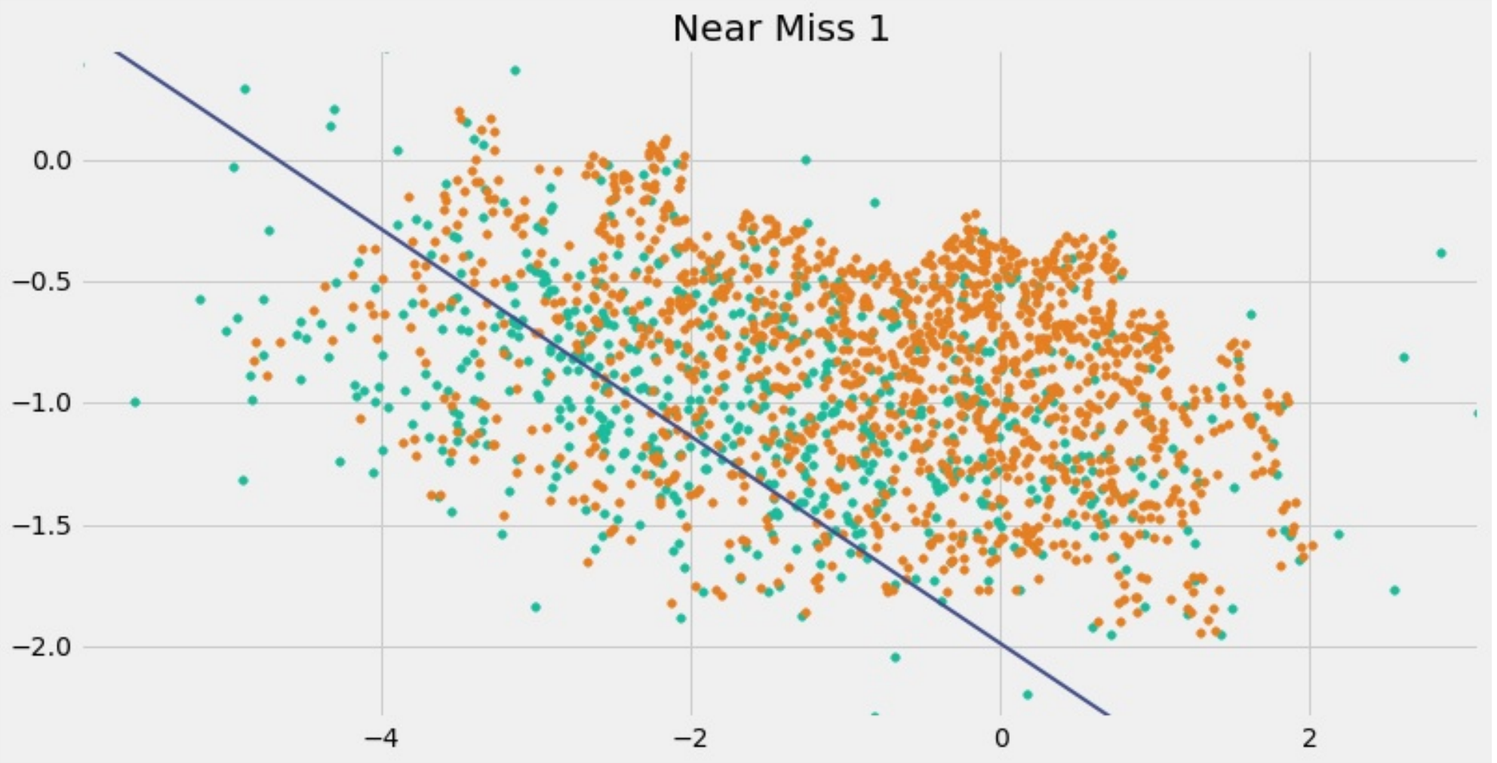
In [20]:

```
# Near Miss 1
us = NearMiss(ratio=0.5, size_ngh=3, version=1, random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 1360, 'S': 680})
```

In [21]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Near Miss 1")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [23]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.92      0.95      0.94      2680
          S       0.44      0.32      0.37       320

avg / total       0.87      0.88      0.88      3000
```

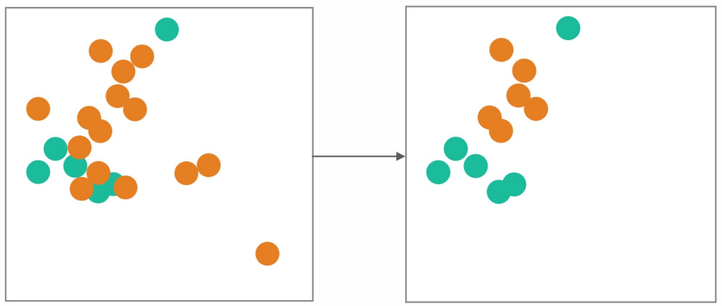
In [24]:
```
precision_recall_comparison
```

Out[24]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82

NEARMISS-2

- Keep points from L whose mean distance to the k farthest points in S is lowest



Mani, Inderjeet, and I. Zhang. "kNN approach to unbalanced data distributions: a case study involving information extraction." Proceedings of workshop on learning from imbalanced datasets. 2003.

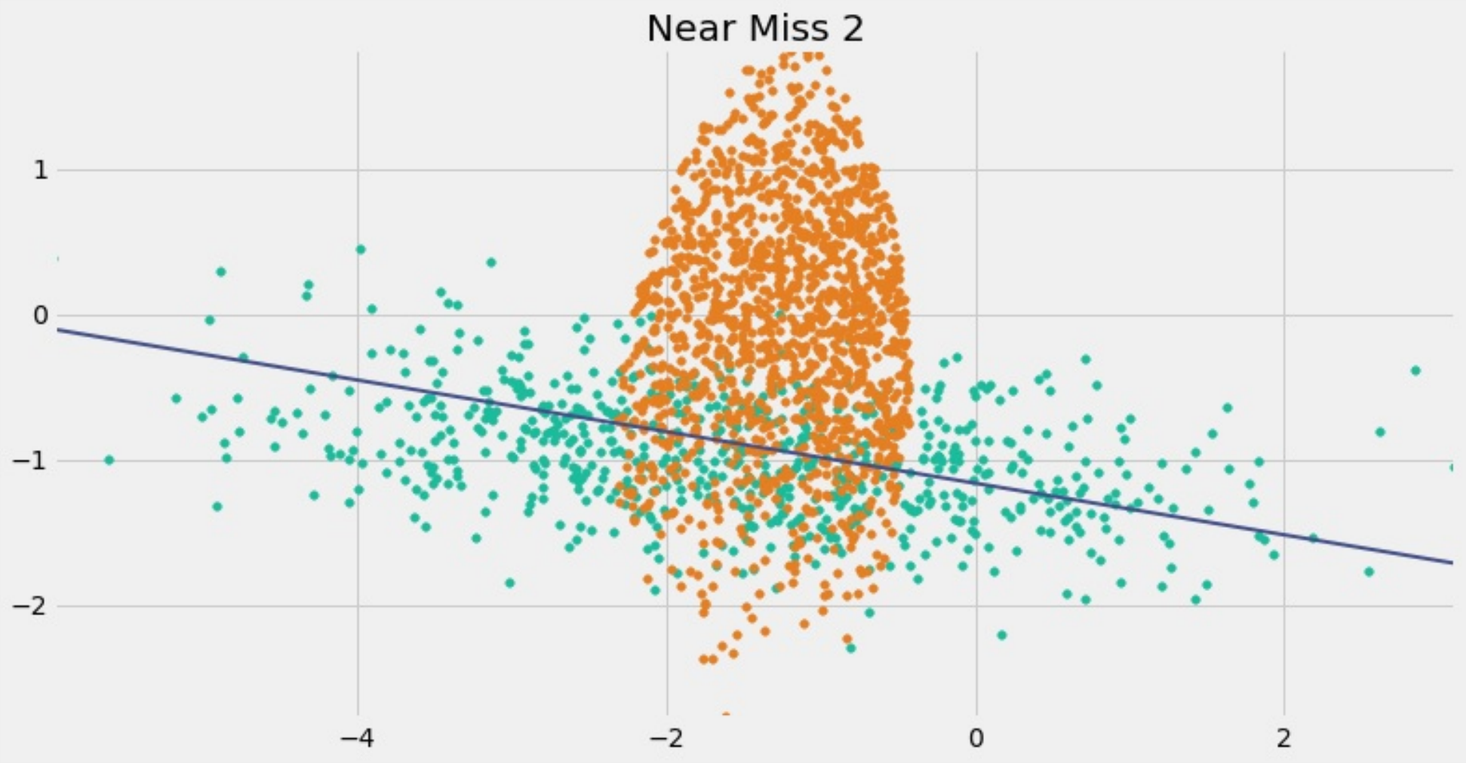
In [25]:

```
# Near Miss 2
us = NearMiss(ratio=0.5, size_ngh=3, version=2, random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 1360, 'S': 680})
```

In [26]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Near Miss 2")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [28]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.95      0.89      0.92      2680
          S       0.39      0.60      0.47       320

avg / total       0.89      0.86      0.87      3000
```

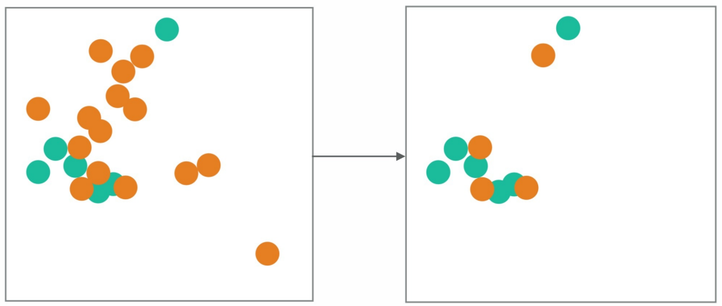
In [29]:
```
precision_recall_comparison
```

Out[29]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32

NEARMISS-3

- Select k nearest neighbors in L for every point in S



Mani, Inderjeet, and I. Zhang. "kNN approach to unbalanced data distributions: a case study involving information extraction." Proceedings of workshop on learning from imbalanced datasets. 2003.

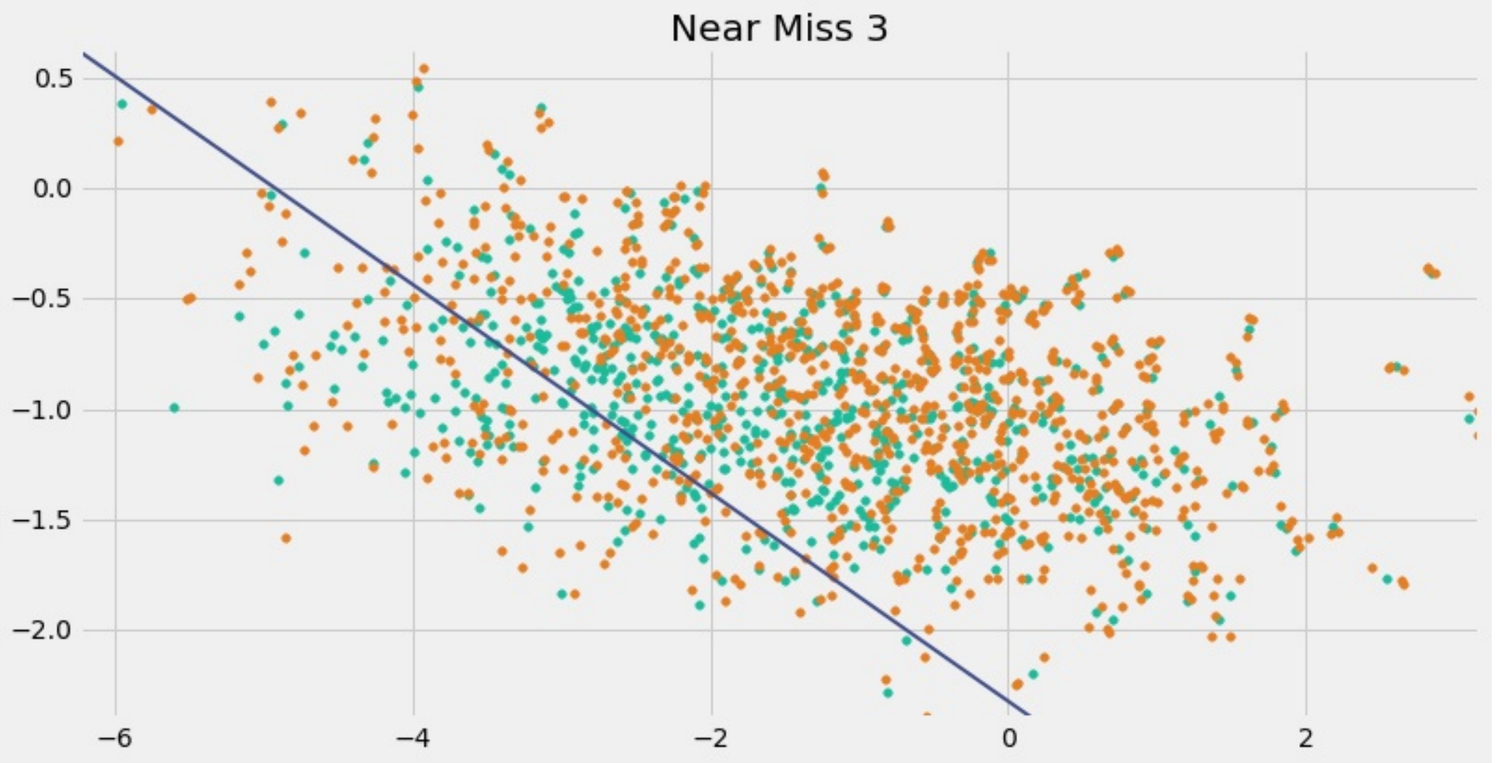
In [30]:

```
# Near Miss 3
us = NearMiss(ratio=0.5, size_ngh=3, ver3_samp_ngh=3, version=3, random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 964, 'S': 680})
```

In [31]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Near Miss 3")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [33]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.91      0.97      0.94      2680
          S       0.43      0.20      0.28       320

avg / total       0.86      0.89      0.87      3000
```

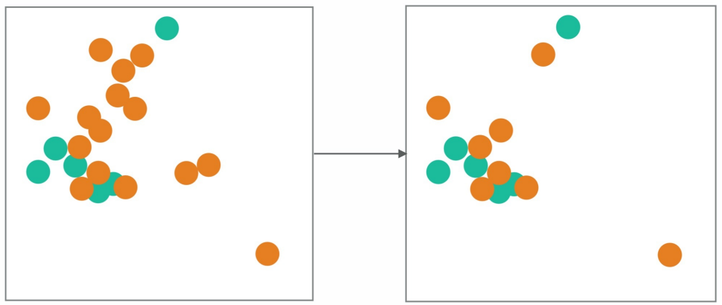
In [34]:
```
precision_recall_comparison
```

Out[34]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60

CONDENSED NEAREST NEIGHBOR (CNN)

- Choose a subset of examples U from the training set T such that for every point in T, its nearest neighbor from U is of the same class
    - Select a random point p from T and set U={p}
    - Scan T-U and add to U the first point found whose nearest neighbor in U is of a different class
    - Repeat the previous step until U is maximal
- Variant: Undersample L using CNN



Hart, P. "The condensed nearest neighbor rule (Corresp.)." IEEE Transactions on Information Theory 14.3 (1968): 515-516.

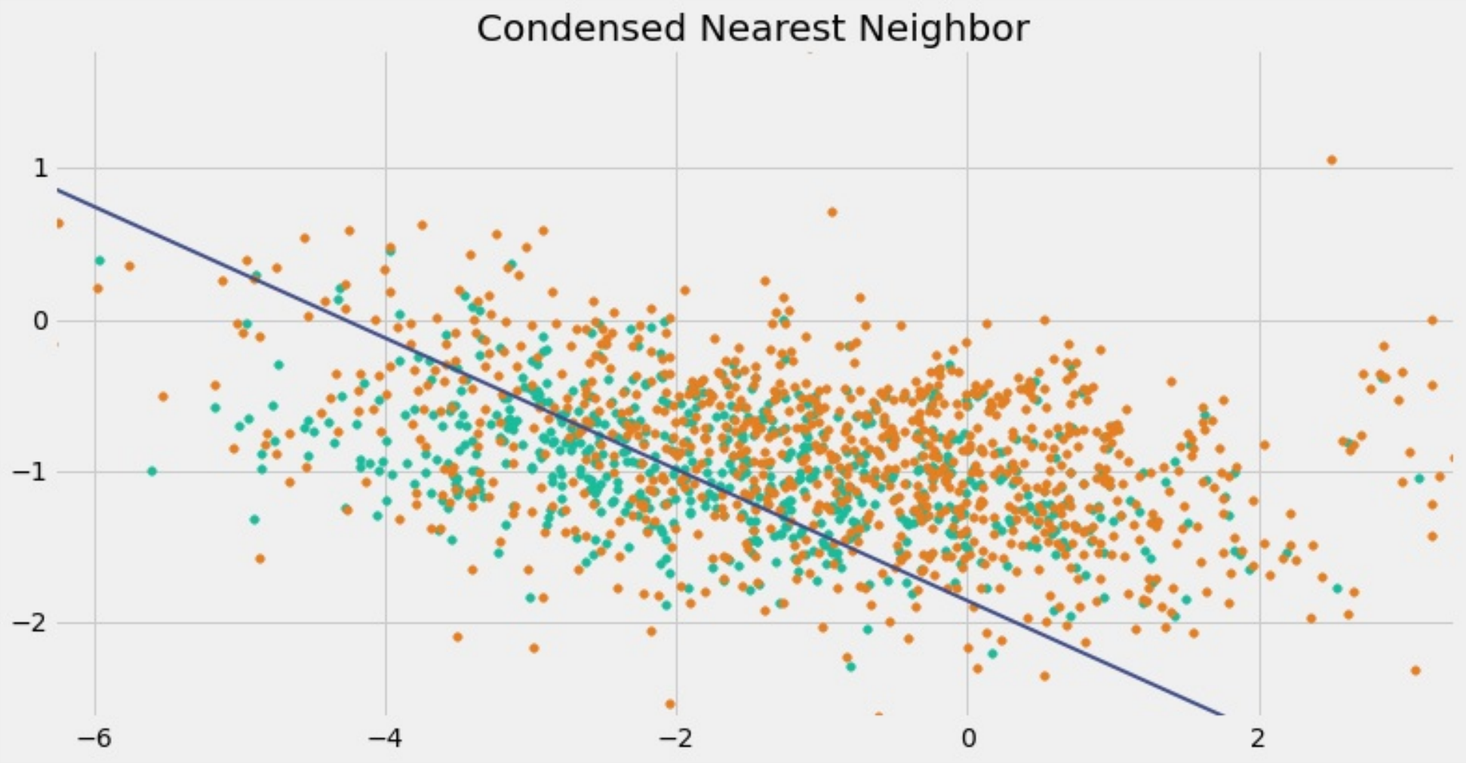
In [35]:

```
# Condensed Nearest Neighbor
us = CondensedNearestNeighbour(random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 882, 'S': 680})
```

In [36]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Condensed Nearest Neighbor")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [38]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.93      0.94      0.93      2680
          S       0.44      0.39      0.42       320

avg / total       0.88      0.88      0.88      3000
```

In [39]:
```
precision_recall_comparison
```

Out[39]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20

EDITED NEAREST NEIGHBOR (ENN)

- Remove examples whose class differs from a majority of its k-nearest neighbors

- A variant is Repeated ENN

    - The above editing is done repeatedly on until the remaining data set is minimal

Wilson, Dennis L. "Asymptotic properties of nearest neighbor rules using edited data." IEEE Transactions on Systems, Man, and Cybernetics 3 (1972): 408-421.

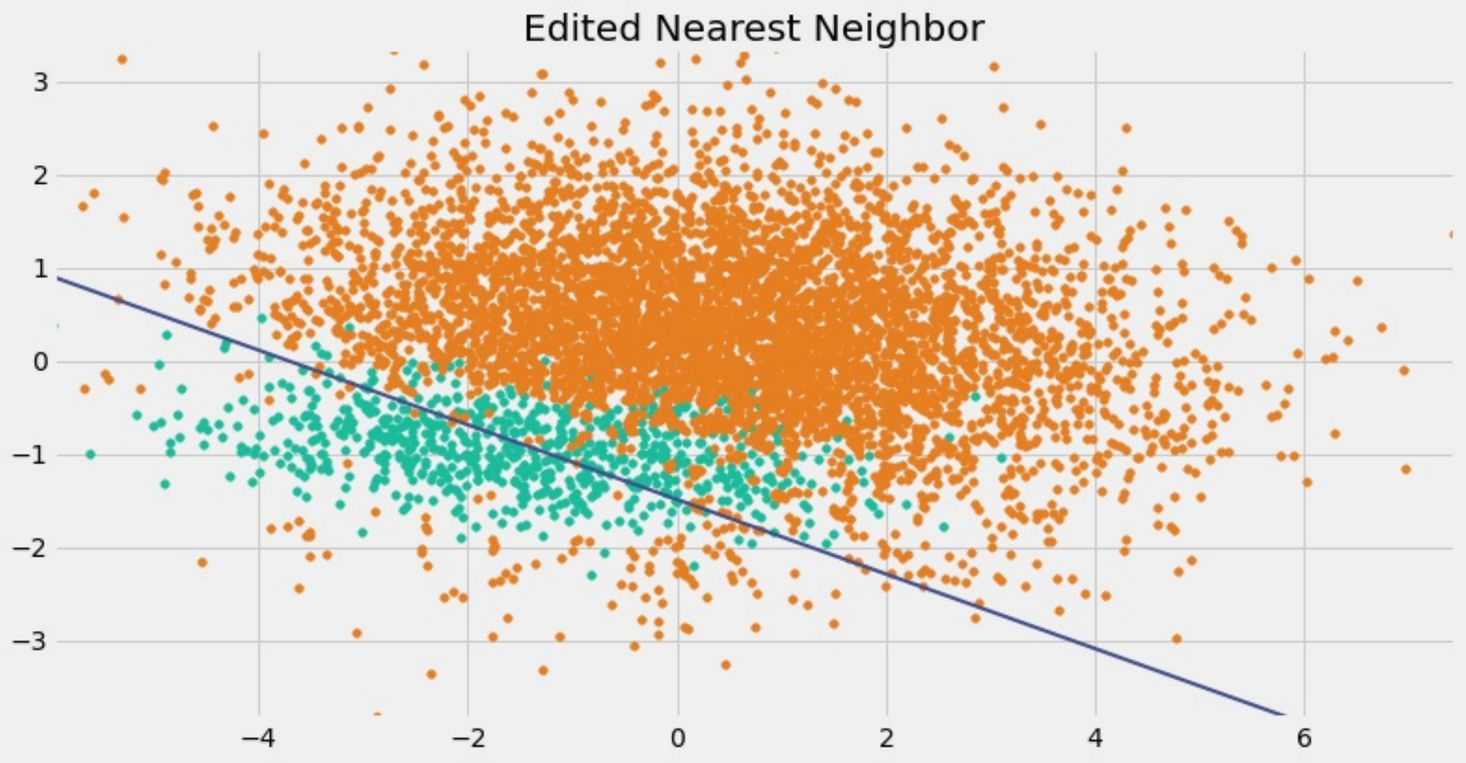
In [40]:

```
# Edited Nearest Neighbor (ENN)
us = EditedNearestNeighbours(size_ngh=5, random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 5120, 'S': 680})
```

In [41]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Edited Nearest Neighbor")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [43]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.95      0.90      0.92      2680
          S       0.42      0.59      0.49       320

avg / total       0.89      0.87      0.88      3000
```

In [44]:
```
precision_recall_comparison
```

Out[44]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39

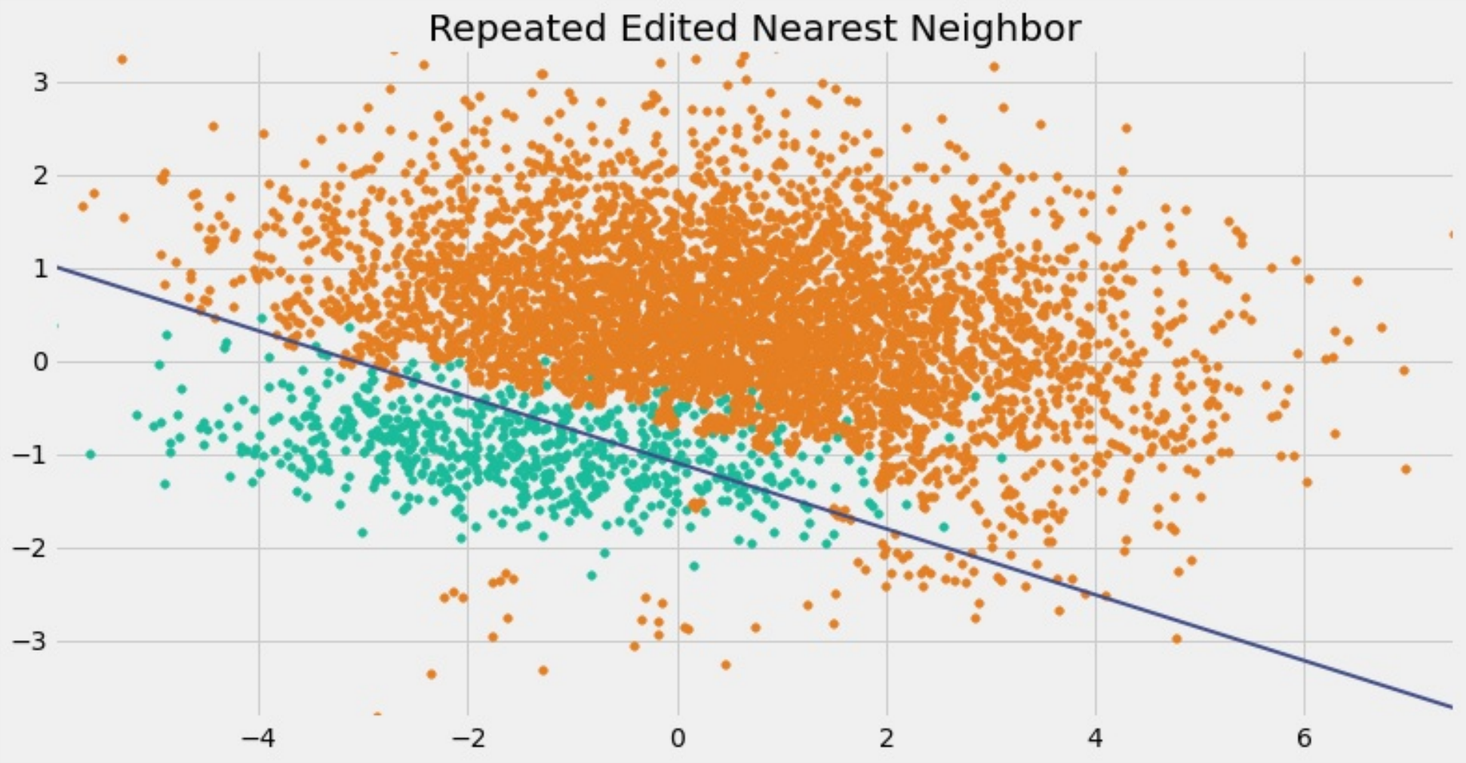
In [45]:

```
# Repeated Edited Nearest Neighbor
us = RepeatedEditedNearestNeighbours(size_ngh=5, random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 4796, 'S': 680})
```

In [46]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Repeated Edited Nearest Neighbor")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [48]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.97      0.84      0.90      2680
          S       0.38      0.80      0.51       320

avg / total       0.91      0.84      0.86      3000
```

In [49]:
```
precision_recall_comparison
```

Out[49]:

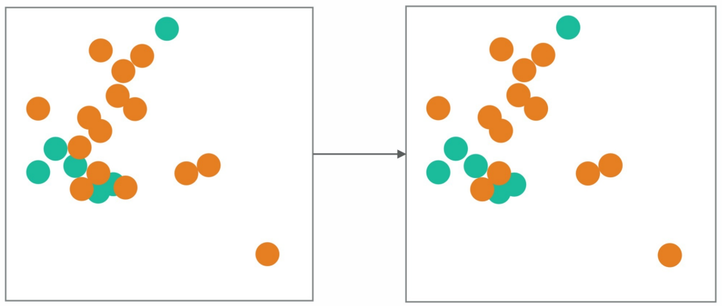
Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59

TOMEK LINK REMOVAL

- A pair of examples is called a Tomek link if they belong to different classes and are each other's nearest neighbors

- Remove all Tomek links from the dataset

- Variant: Remove Tomek link members from L



Tomek, Ivan. "Two modifications of CNN." IEEE Trans. Systems, Man and Cybernetics 6 (1976): 769-772.

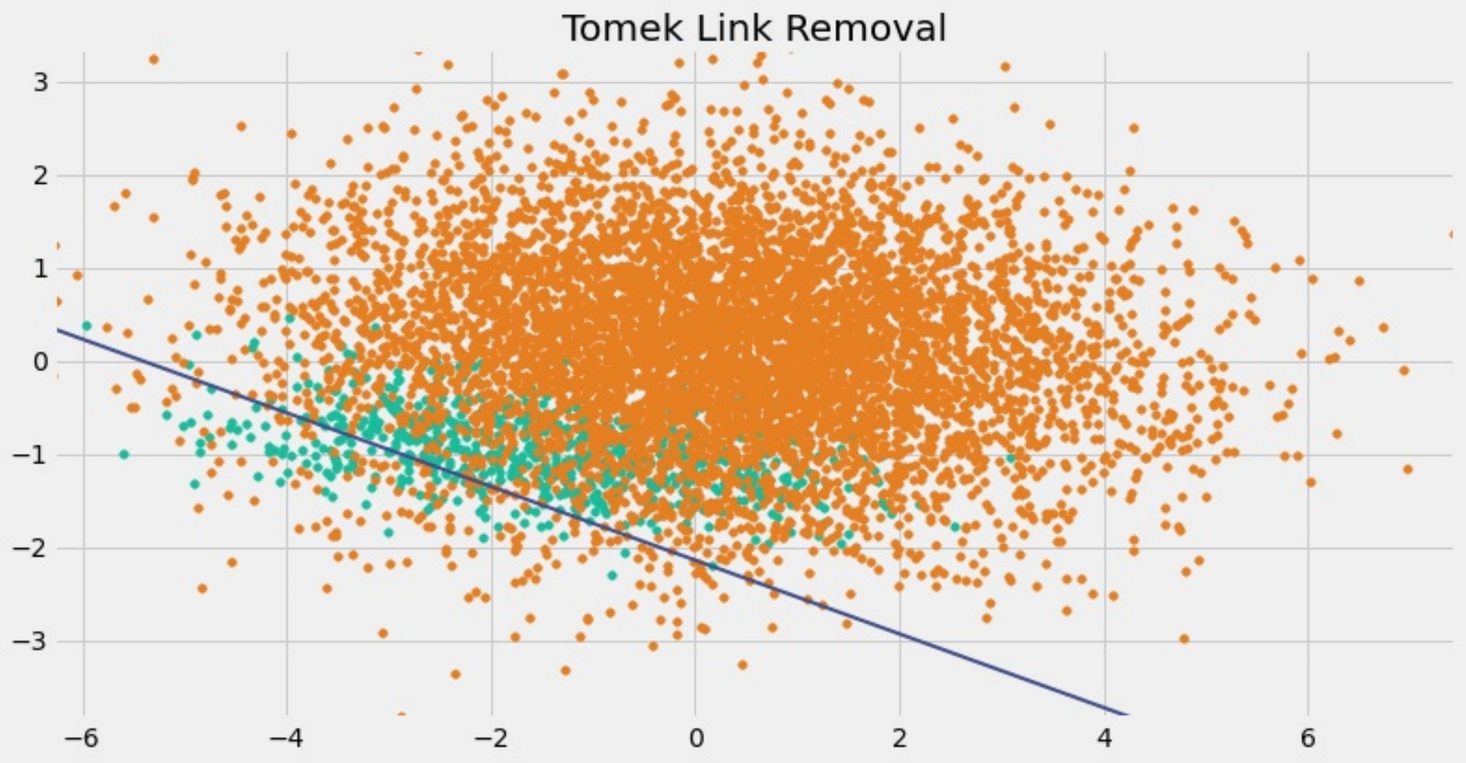
In [50]:

```
# Tomek link removal
us = TomekLinks(random_state=1)
X_train_res, y_train_res = us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 6051, 'S': 680})
```

In [51]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Tomek Link Removal")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [53]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.91      0.97      0.94      2680
          S       0.44      0.21      0.29       320

avg / total       0.86      0.89      0.87      3000
```

In [54]:
```
precision_recall_comparison
```

Out[54]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80

RANDOM OVERSAMPLING

- Random oversampling of minority class
- May lead to overfitting

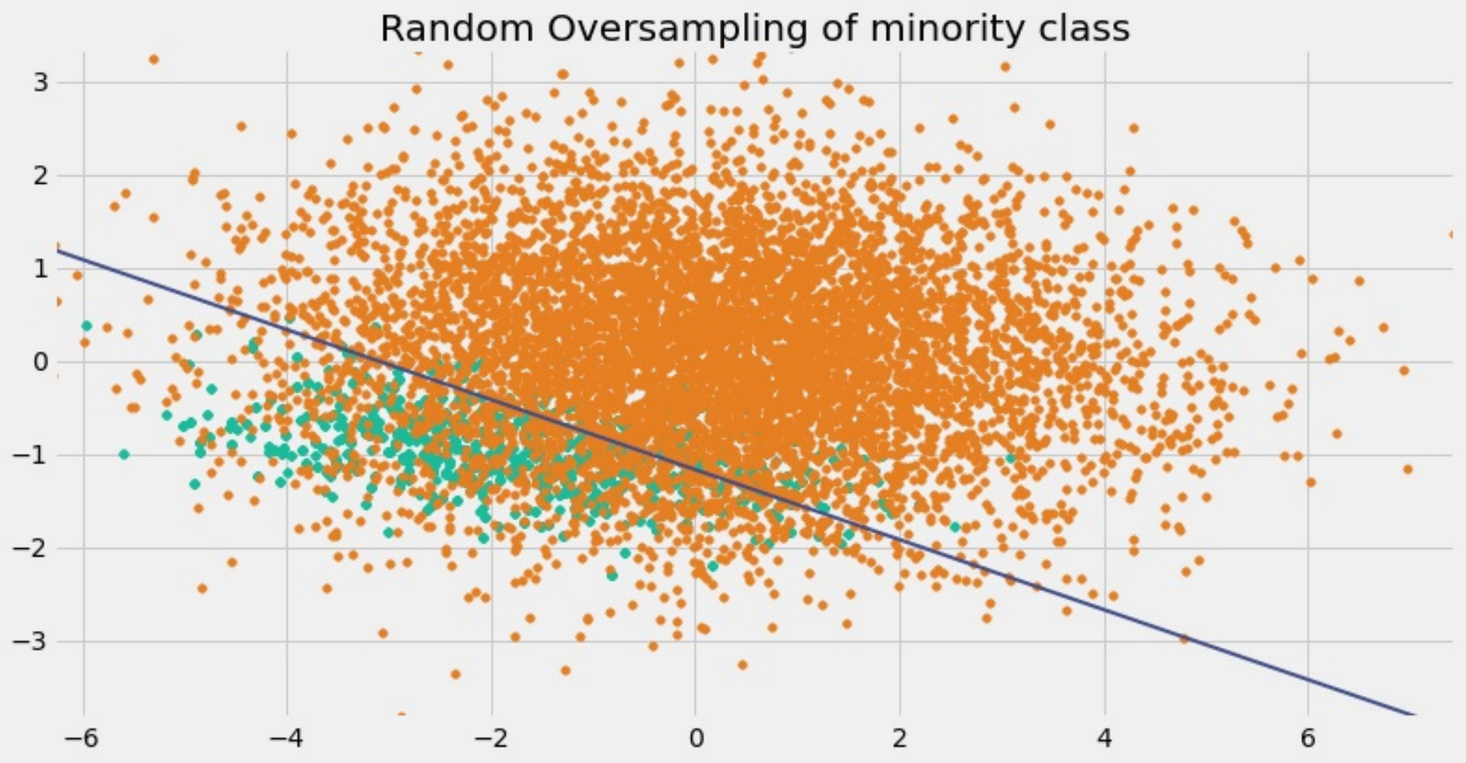
In [55]:

```
# Random oversampling of minority class
os = RandomOverSampler(ratio=0.5, random_state=1)
X_train_res, y_train_res = os.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 6320, 'S': 3160})
```

In [56]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("Random Oversampling of minority class")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [58]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.97      0.85      0.91      2680
          S       0.38      0.76      0.51       320

avg / total       0.91      0.84      0.86      3000
```

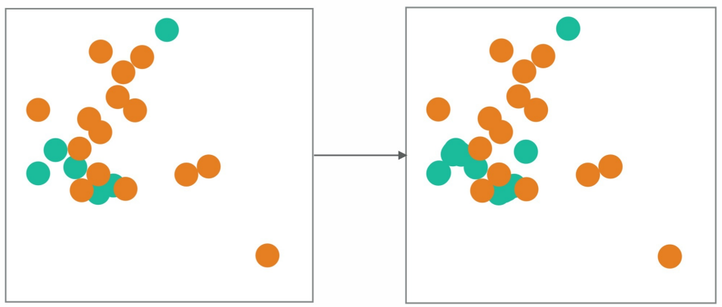
In [59]:
```
precision_recall_comparison
```

Out[59]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80
Tomek Link Removal	0.91	0.21

SYNTHETIC MINORITY OVERSAMPLING TECHNIQUE (SMOTE)

- For each point p in S:
  - Compute its k nearest neighbors in S
  - Randomly choose r≤k of the neighbors (with replacement)
  - Choose a random point along each of the lines joining p and each of the r neighbors
  - Add these synthetic points to the dataset with class S



Chawla, Nitesh V., et al. "SMOTE: synthetic minority over-sampling technique." Journal of artificial intelligence research 16 (2002): 321-357.

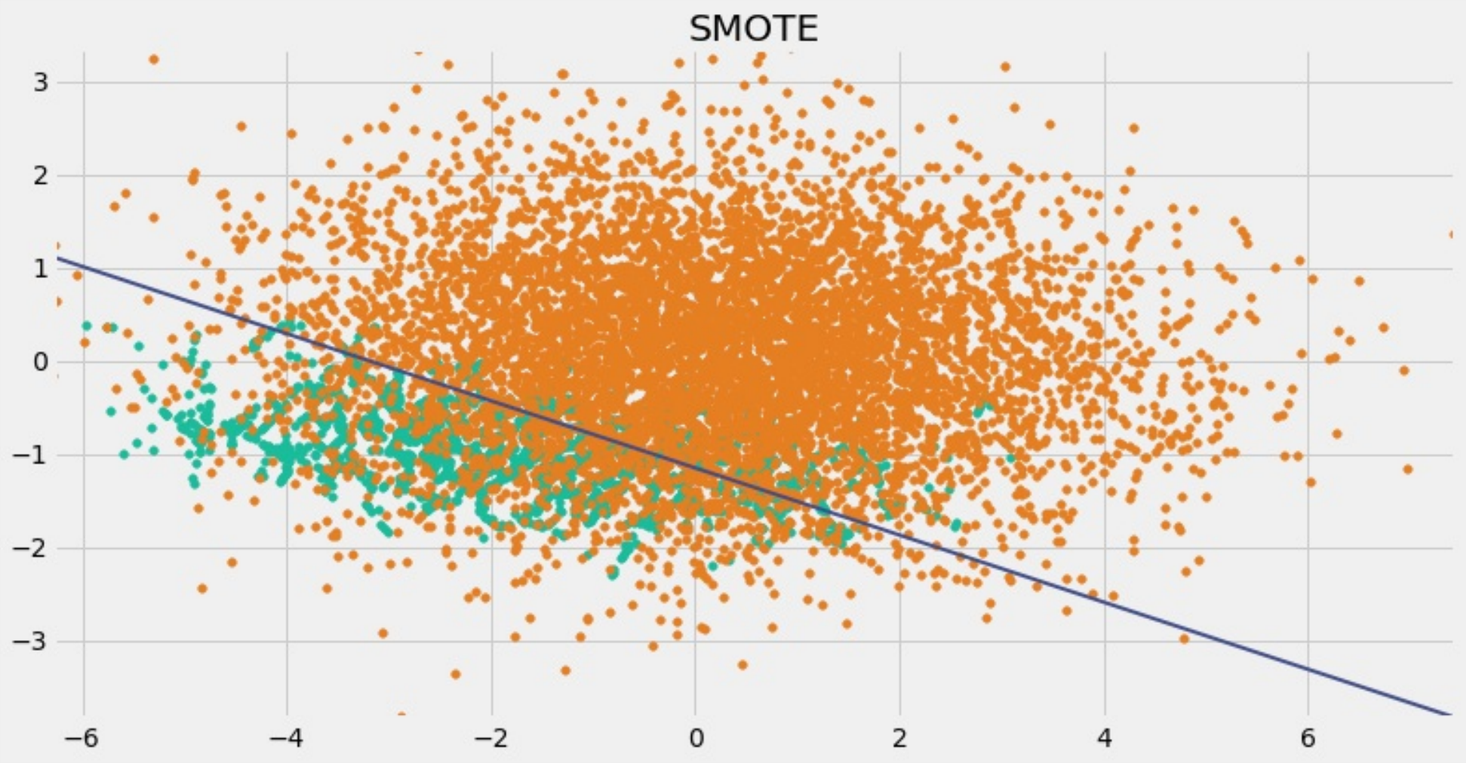
In [60]:

```
# Synthetic Minority Oversampling Technique (SMOTE)
os = SMOTE(ratio=0.5, k=5, random_state=1)
X_train_res, y_train_res = os.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 6320, 'S': 3160})
```

In [61]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("SMOTE")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [63]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.97      0.85      0.91      2680
          S       0.38      0.77      0.51       320

avg / total       0.91      0.84      0.86      3000
```

In [64]:
```
precision_recall_comparison
```

Out[64]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80
Tomek Link Removal	0.91	0.21
Random oversampling of S	0.97	0.76

COMBINATIONS

- SMOTE+Tomek Link Removal: Oversampling using SMOTE followed by undersampling using Tomek Link Removal

- SMOTE+ENN: Oversampling using SMOTE followed by undersampling using ENN

Batista, Gustavo EAPA, Ronaldo C. Prati, and Maria Carolina Monard. "A study of the behavior of several methods for balancing machine learning training data." ACM Sigkdd Explorations Newsletter 6.1 (2004): 20-29.

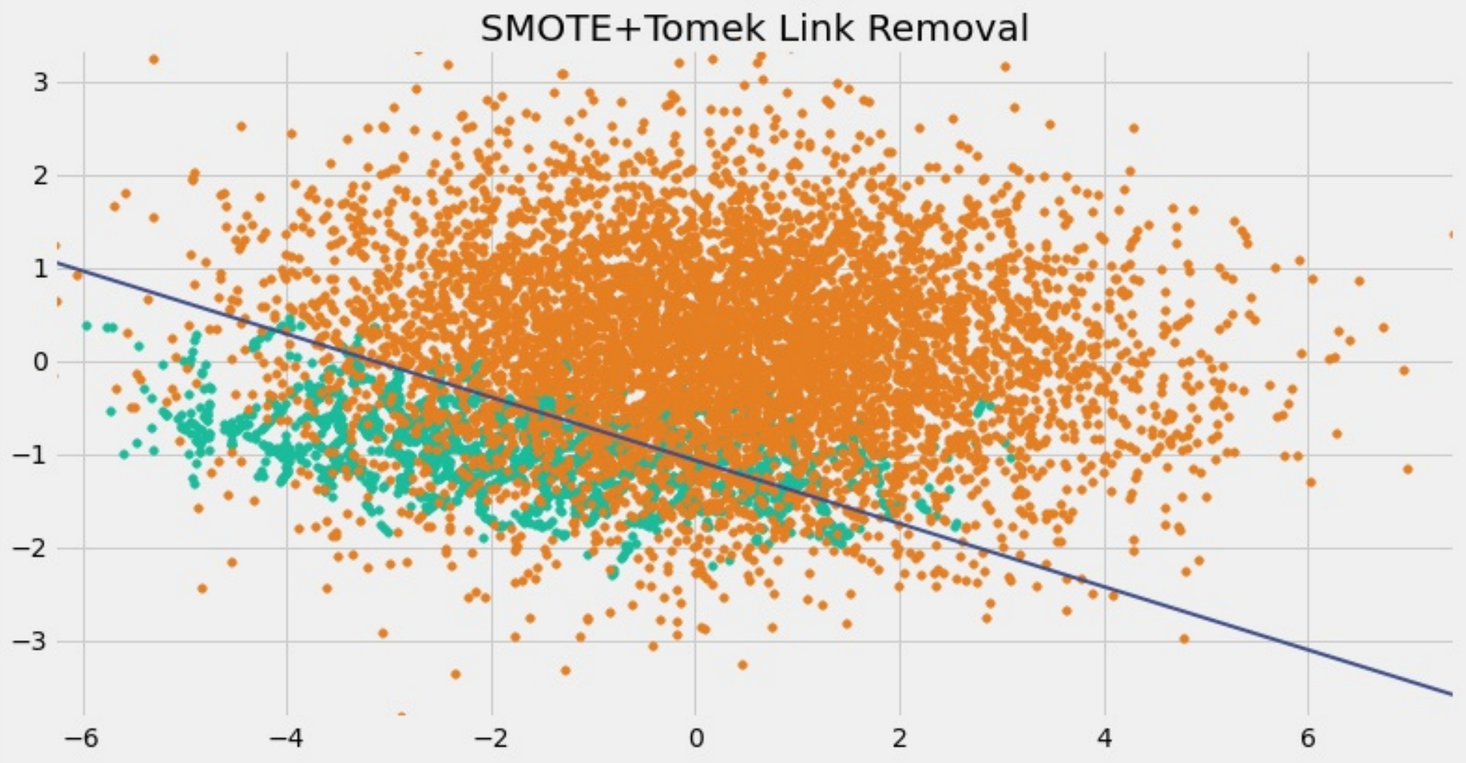
In [65]:

```
# SMOTE + Tomek link removal
os_us = SMOTETomek(ratio=0.5, k=5, random_state=1)
X_train_res, y_train_res = os_us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 6050, 'S': 3160})
```

In [66]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("SMOTE+Tomek Link Removal")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [68]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.97      0.84      0.90      2680
          S       0.38      0.80      0.51       320

avg / total       0.91      0.84      0.86      3000
```

In [69]:
```
precision_recall_comparison
```

Out[69]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80
Tomek Link Removal	0.91	0.21
Random oversampling of S	0.97	0.76
SMOTE	0.97	0.77

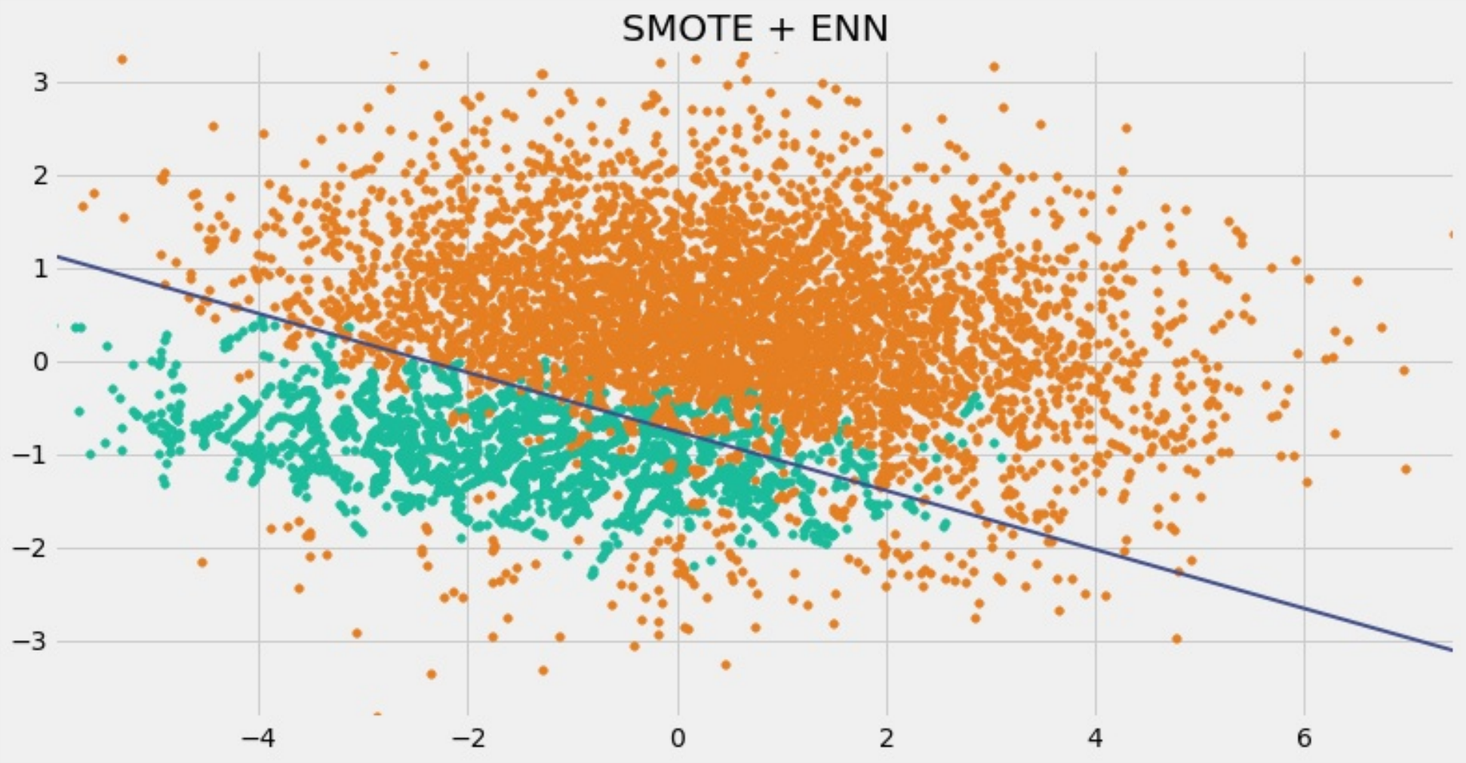
In [70]:

```
# SMOTE + ENN
os_us = SMOTEENN(ratio=0.5, k=5, size_ngh=5, random_state=1)
X_train_res, y_train_res = os_us.fit_sample(X_train, y_train)

print "Distribution of class labels before resampling {}".format(Counter(y_train))
print "Distribution of class labels after resampling {}".format(Counter(y_train_re
s))

clf_base = LogisticRegression()
grid = {'C': 10.0 ** np.arange(-2, 3),
        'penalty': ['l1', 'l2']}

cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')

clf.fit(X_train_res, y_train_res)

coef = clf.best_estimator_.coef_
intercept = clf.best_estimator_.intercept_

x1 = np.linspace(-8, 10, 100)
x2 = -(coef[0][0] * x1 + intercept[0]) / coef[0][1]

X_1_res = X_train_res[y_train_res=='S']
X_2_res = X_train_res[y_train_res=='L']
```

```
Distribution of class labels before resampling Counter({'L': 6320, 'S': 680})
Distribution of class labels after resampling Counter({'L': 4894, 'S': 3160})
```

In [71]:

```
plt.scatter(zip(*X_1_res)[0], zip(*X_1_res)[1], color='#1abc9c')
plt.scatter(zip(*X_2_res)[0], zip(*X_2_res)[1], color='#e67e22')

x_coords = zip(*X_1_res)[0] + zip(*X_2_res)[0]
y_coords = zip(*X_1_res)[1] + zip(*X_2_res)[1]
plt.axis([min(x_coords), max(x_coords), min(y_coords), max(y_coords)])

plt.title("SMOTE + ENN")
plt.plot(x1, x2, color='#414e8a', linewidth=2)
plt.show()
```



In [73]:
```
print classification_report(y_test, clf.predict(X_test))
```

```
             precision    recall  f1-score   support

          L       0.99      0.78      0.87      2680
          S       0.33      0.92      0.49       320

avg / total       0.92      0.79      0.83      3000
```

In [74]:
```
precision_recall_comparison
```

Out[74]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80
Tomek Link Removal	0.91	0.21
Random oversampling of S	0.97	0.76
SMOTE	0.97	0.77
SMOTE + Tomek Link Removal	0.97	0.80

EASYENSEMBLE

- Sample a subset L_i of L such that $|L_i|=|S|$
- Learn an AdaBoost ensemble using L_i and S

$$F_i(x) = sgn(\Sigma_{j=1}^{n_i} w_{ij} f_{ij}(x))$$

- Repeat for N iterations and output a meta ensemble

$$F(x) = sgn(\Sigma_{i=1}^{N} \Sigma_{j=1}^{n_i} w_{ij} f_{ij}(x))$$

Liu, Xu-Ying, Jianxin Wu, and Zhi-Hua Zhou. "Exploratory undersampling for class-imbalance learning." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39.2 (2009): 539-550.

In [75]:

```
# EasyEnsemble
ens = EasyEnsemble()
X_train_res, y_train_res = ens.fit_sample(X_train, y_train)

y_pred_proba = np.zeros(len(y_test))
for idx in range(len(y_train_res)):
    clf_base = AdaBoostClassifier()
    grid = {'n_estimators': [10, 50, 100]}

    cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
    clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')
    clf.fit(X_train_res[idx], y_train_res[idx])
    y_pred_proba += zip(*clf.predict_proba(X_test))[0]

y_pred_proba = y_pred_proba/len(y_train_res)
y_pred = (y_pred_proba > 0.5).astype(int)
y_pred = y_pred.astype('str')
y_pred[y_pred=='1'] ='L'
y_pred[y_pred=='0'] ='S'
```

In [77]:
```
print classification_report(y_test, y_pred)
```

```
             precision    recall  f1-score   support

          L       0.98      0.80      0.88      2680
          S       0.35      0.90      0.50       320

avg / total       0.92      0.81      0.84      3000
```

In [78]:
```
precision_recall_comparison
```

Out[78]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80
Tomek Link Removal	0.91	0.21
Random oversampling of S	0.97	0.76
SMOTE	0.97	0.77
SMOTE + Tomek Link Removal	0.97	0.80
SMOTE + ENN	0.99	0.92

BALANCECASCADE

- Sample a subset L_i of L such that $|L_i|=|S|$
- Learn an AdaBoost ensemble using L_i and S

$$F_i(x) = sgn(\Sigma_{j=1}^{n_i} w_{ij} f_{ij}(x))$$

- Remove all examples from L that are correctly classified by F_i
- Repeat for N iterations and combine

$$F(x) = sgn(\Sigma_{i=1}^{N} \Sigma_{j=1}^{n_i} w_{ij} f_{ij}(x))$$

Liu, Xu-Ying, Jianxin Wu, and Zhi-Hua Zhou. "Exploratory undersampling for class-imbalance learning." IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 39.2 (2009): 539-550.

In [79]:

```
# BalanceCascade

# Note this is not a true implementation of BalancaCacade since the library
# does not return the fitted classifiers used in generating the subsets
ens = BalanceCascade(classifier='adaboost', random_state=1)
X_train_res, y_train_res = ens.fit_sample(X_train, y_train)

y_pred_proba = np.zeros(len(y_test))
for idx in range(len(y_train_res)):
    # imblearn uses AdaBoostClassiifer with default hyperparams to select subsets
    # a variant implementation with grid search for each subsample is shown in the
 next cell
    clf = AdaBoostClassifier(random_state=1)
    clf.fit(X_train_res[idx], y_train_res[idx])
    y_pred_proba += zip(*clf.predict_proba(X_test))[0]

y_pred_proba = y_pred_proba/len(y_train_res)
y_pred = (y_pred_proba > 0.5).astype(int)
y_pred = y_pred.astype('str')
y_pred[y_pred=='1'] ='L'
y_pred[y_pred=='0'] ='S'
```

In [81]:
```
print classification_report(y_test, y_pred)
```

```
             precision    recall  f1-score   support

          L       0.99      0.81      0.89      2680
          S       0.36      0.91      0.51       320

avg / total       0.92      0.82      0.85      3000
```

In [82]:
```
precision_recall_comparison
```

Out[82]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80
Tomek Link Removal	0.91	0.21
Random oversampling of S	0.97	0.76
SMOTE	0.97	0.77
SMOTE + Tomek Link Removal	0.97	0.80
SMOTE + ENN	0.99	0.92
EasyEnsemble	0.98	0.87

In [83]:

```
# Note this is not a true implementation of BalancaCacade since the library
# does not return the fitted classifiers used in generating the subsets
ens = BalanceCascade(classifier='adaboost', random_state=1)
X_train_res, y_train_res = ens.fit_sample(X_train, y_train)

y_pred_proba = np.zeros(len(y_test))
for idx in range(len(y_train_res)):
    clf_base = AdaBoostClassifier(random_state=1)
    grid = {'n_estimators': [10, 50, 100]}
    cv = KFold(X_train_res.shape[0], n_folds=5, shuffle=True, random_state=0)
    clf = GridSearchCV(clf_base, grid, cv=cv, n_jobs=8, scoring='f1_macro')
    clf.fit(X_train_res[idx], y_train_res[idx])
    y_pred_proba += zip(*clf.predict_proba(X_test))[0]

y_pred_proba = y_pred_proba/len(y_train_res)
y_pred = (y_pred_proba > 0.5).astype(int)
y_pred = y_pred.astype('str')
y_pred[y_pred=='1'] ='L'
y_pred[y_pred=='0'] ='S'
```

In [85]:
```
print classification_report(y_test, y_pred)
```

```
             precision    recall  f1-score   support

          L       0.99      0.80      0.88      2680
          S       0.35      0.90      0.50       320

avg / total       0.92      0.81      0.84      3000
```

In [86]:
```
precision_recall_comparison
```

Out[86]:

Method	Precision on L	Recall on S
Baseline	0.90	0.12
Weighted classes	0.98	0.89
Random undersampling of L	0.97	0.82
NearMiss-1	0.92	0.32
NearMiss-2	0.95	0.60
NearMiss-3	0.91	0.20
CNN	0.93	0.39
ENN	0.95	0.59
Repeated ENN	0.97	0.80
Tomek Link Removal	0.91	0.21
Random oversampling of S	0.97	0.76
SMOTE	0.97	0.77
SMOTE + Tomek Link Removal	0.97	0.80
SMOTE + ENN	0.99	0.92
EasyEnsemble	0.98	0.87
BalanceCascade	0.99	0.91

EPILOGUE

- No free lunch!
- Different methods work well with different data distributions, classifiers and objectives
- Other techniques: Borderline SMOTE, SMOTEBoost, ADA-SYN, CBO, AdaC1, AdaC2, AdaC3, Kernel based methods, Active learning,…
- Within class vs between class imbalance

He, Haibo, and Edwardo A. Garcia. "Learning from imbalanced data." IEEE Transactions on knowledge and data engineering 21.9 (2009): 1263-1284.

THANK YOU!

ajinkya.more@gmail.com