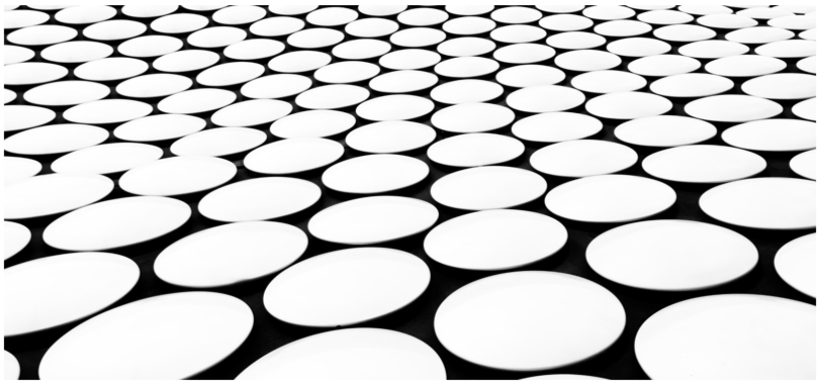


SCHOOL OF AI ITALIA

MACHINE LEARNING AVANZATO DA ZERO

ANTONIO DI CECCO - SCHOOL OF AI

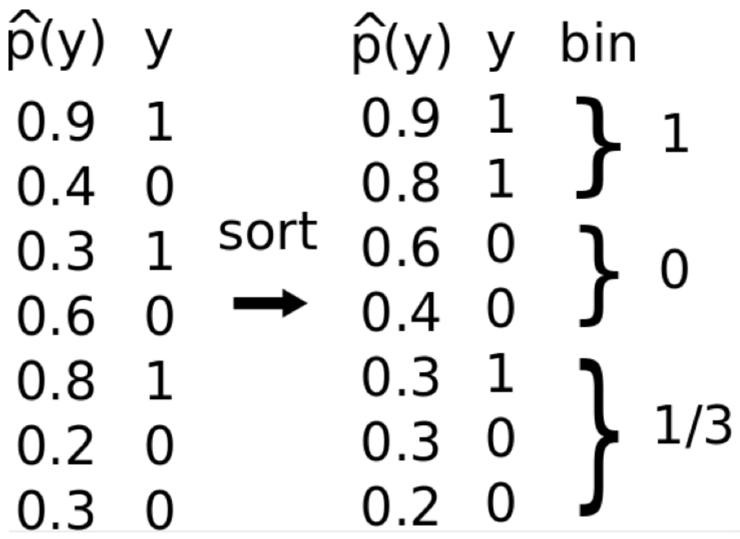
Dataset sbilanciati: G1no, SMOTE (e calibrazione)

Calibrazione

Source

- Probabilities can be much more informative than labels:
- “The model predicted you don’t have cancer” vs “The model predicted you’re 40% likely to have cancer”

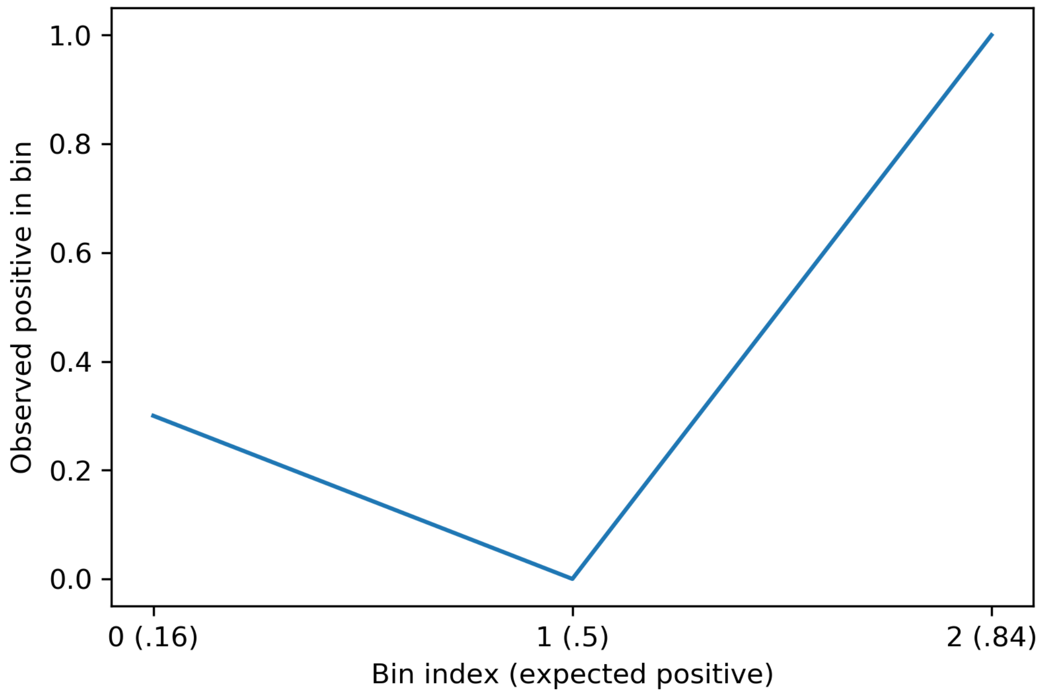
Curva di calibrazione (diagramma di reliability)



$\hat{p}(y)$	y
0.9	1
0.4	0
0.3	1
0.6	0
0.8	1
0.2	0
0.3	0

$\hat{p}(y)$	y	bin
0.9	1	1
0.8	1	
0.6	0	0
0.4	0	
0.3	1	1/3
0.3	0	
0.2	0	

riordiniamo!!!



vorrei una diagonale

- For binary classification only
- you can be calibrated and inaccurate!
- Given a predicted ranking or probability from a supervised classifier, bin predictions.
- Plot fraction of data that’s positive in each bin.
- Doesn’t require ground truth probabilities!

If these probabilities estimates were actually accurate, then let's say if I have a bin for all the 90% ones, the prevalence in the 90% bin should actually be 90%, that's sort of what it says. 90% of the points that are given a score of 90% should be the true class. And so you can plot this in the calibration curve or the reliability diagram.

So here I have three bins. And basically what I wanted is the diagonal line where the bin that contains the very small probabilities has the same prevalence. And so here, for example, things that are around like 0.5 actually have zero prevalence. So there's no true class in there, which is really weird. You won’t use a diagonal line, this would be a very bad classifier.

One thing that I want to emphasize here is that calibration doesn't imply that the model is accurate. These are actually two different things. If I have a binary model on a balanced data set, and it always says 50% probability for all data points. It's perfectly calibrated because it says 50% for all data points. 50% of those data points are actually the true class if the dataset is balanced. So it's a completely trivial classifier that doesn't tell you anything but perfectly calibrated, but also kind of tells you nothing because it always gives you 0.5. So you know that you can trust it basically.

Calibration basically tells you how much you can trust the model.

- For binary classification only
- you can be calibrated and inaccurate!
- Given a predicted ranking or probability from a supervised classifier, bin predictions.
- Plot fraction of data that’s positive in each bin.
- Doesn’t require ground truth probabilities!

Calibration_curve con sklearn

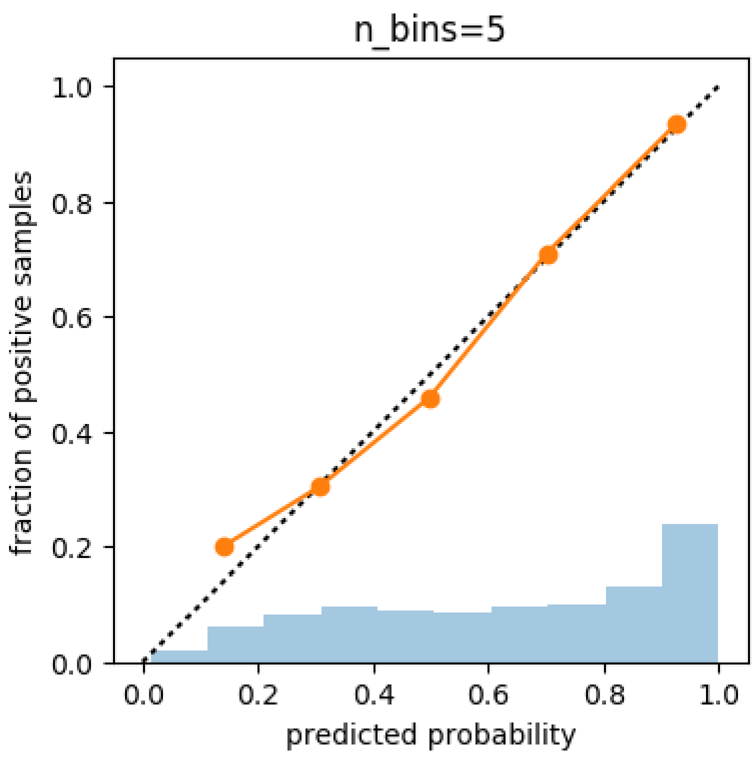
Using subsample of dataset

```
from sklearn.linear_model import LogisticRegressionCV
print(X_train.shape)
print(np.bincount(y_train))
lr = LogisticRegressionCV().fit(X_train, y_train)
(52292, 54)
[19036 33256]
```

```
lr.C_
array([ 2.783])
```

```
print(lr.predict_proba(X_test)[:10])
print(y_test[:10])
[[ 0.681  0.319]
 [ 0.049  0.951]
 [ 0.706  0.294]
 [ 0.537  0.463]
 [ 0.819  0.181]
 [ 0.     1.   ]
 [ 0.794  0.206]
 [ 0.676  0.324]
 [ 0.727  0.273]
 [ 0.597  0.403]]
[0 1 0 1 1 1 0 0 0 1]
```

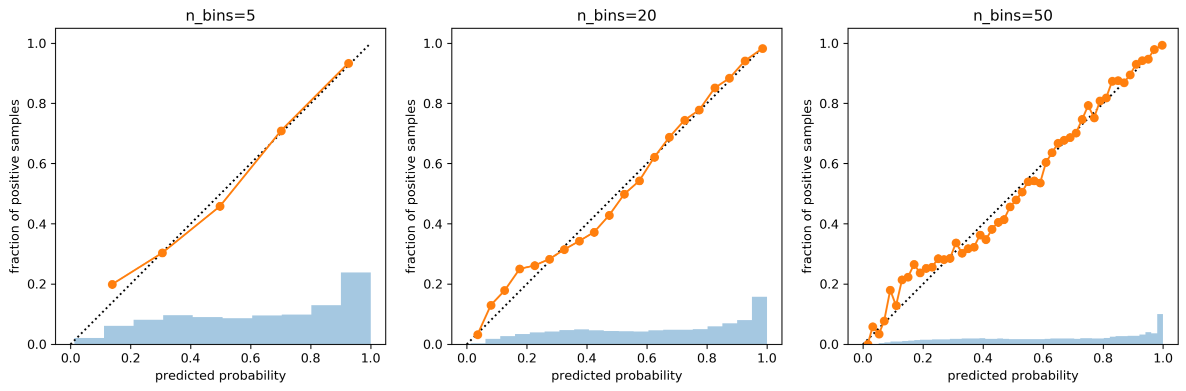
```
from sklearn.calibration import calibration_curve
probs = lr.predict_proba(X_test)[:, 1]
prob_true, prob_pred = calibration_curve(y_test, probs, n_bins=5)
print(prob_true)
print(prob_pred)
[ 0.2    0.303  0.458  0.709  0.934]
[ 0.138  0.306  0.498  0.701  0.926]
```



ovviamente su una validazione

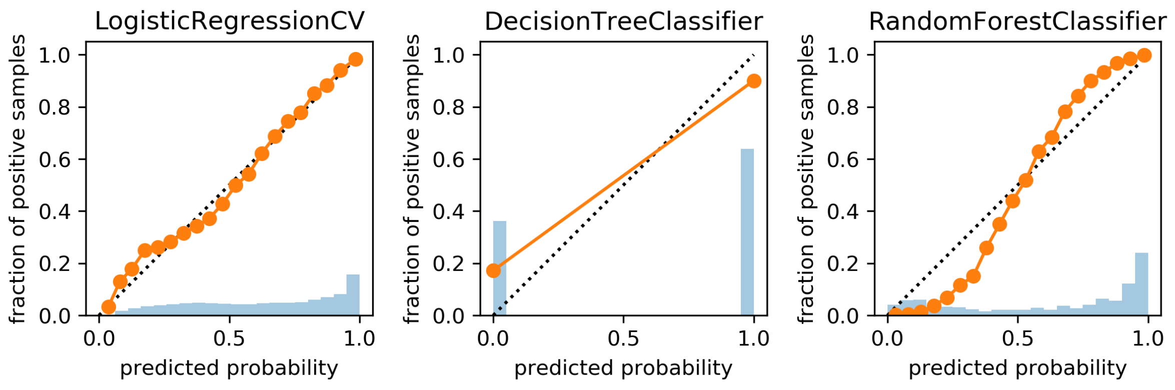
Logistic regression typically gives pretty good probability estimates.

Influenza del numero di bin



- Works here because dataset is big
- Usate la CFD empirica che è meglio

Paragoniamo i modelli



Logistic Regression OK

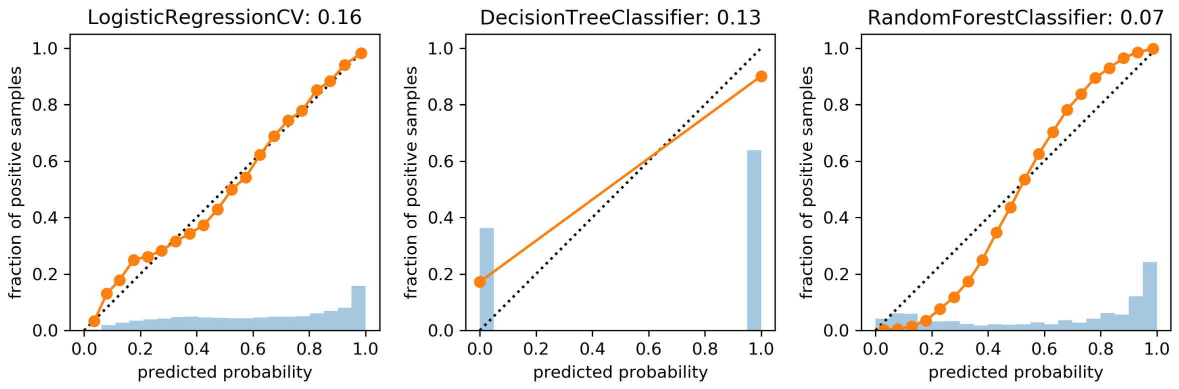
DecisionTree senza pruning un po’ troppo sicuro

Random Forest non è sufficientemente certo

Brier Score (per la classificazione binaria)

- "mean squared error of probability estimate"

$$BS = \frac{\sum_{i=1}^{n} (\hat{p}(y_i) - y_i)^2}{n}$$



y_i o è 1 o 0

Sistemiamolo: calibriamo un classificatore

- Build another model, mapping classifier probabilities to better probabilities!
- 1d model! (or more for multi-class)

  $f_{calib}(s(x)) \approx p(y)$
- s(x) is score given by model, usually
- Can also work with models that don't even provide probabilities! Need model for f_{calib}, need to decide what data to train it on.
- Can train on training set $\rightarrow$ Overfit
- Can train using cross-validation $\rightarrow$ use data, slower

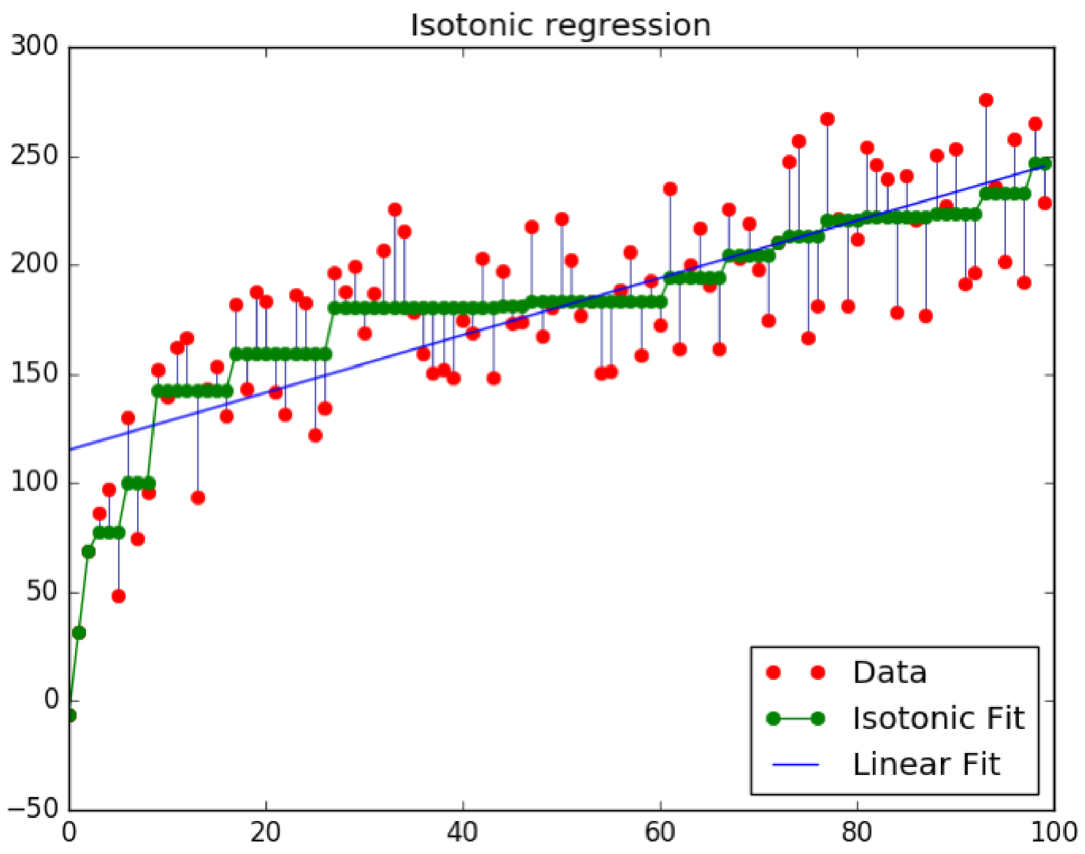
Platt Scaling

- Usiamo una logistica, una sigmoide per f_{calib}
- Basically learning a 1d logistic regression (+ some tricks)
- Funziona bene per la SVM

$$f_{platt} = \frac{1}{1 + exp(-ws(x) - b)}$$

Isotonic Regression

- molto flessivile e universale f_{calib}
- Learns arbitrary monotonically increasing step-functions in 1d.
- Groups data into constant parts, steps in between.
- Optimum monotone function on training data (wrt mse).



Maurizio Parton Alpha Go con handicap... logistic regression ovvero il Platt Scaling... Isotonic Fit

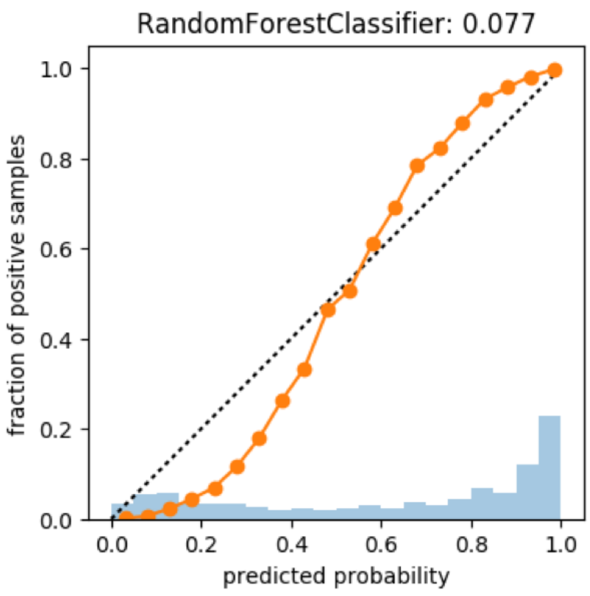
Costruire il modello

- Non usate il training set
- quindi o un validation set o cross-validation

Fittiamo il modello di Calibrazione

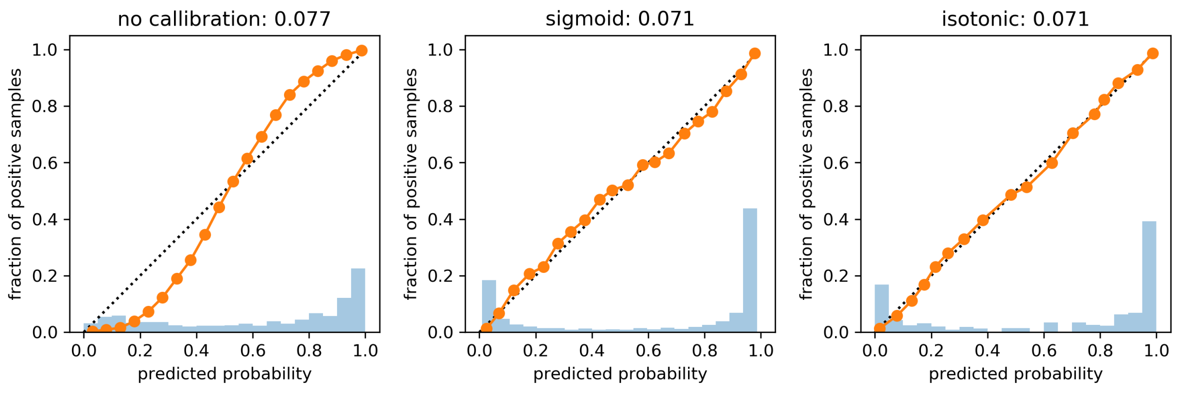
usiamo CalibratedClassifierCV

```
from sklearn.calibration import CalibratedClassifierCV
X_train_sub, X_val, y_train_sub, y_val = train_test_split(X_train, y_train,
                                                          stratify=y_train,
random_state=0)
rf = RandomForestClassifier().fit(X_train_sub, y_train_sub)
scores = rf.predict_proba(X_test)[:, 1]
plot_calibration_curve(y_test, scores, n_bins=20)
```

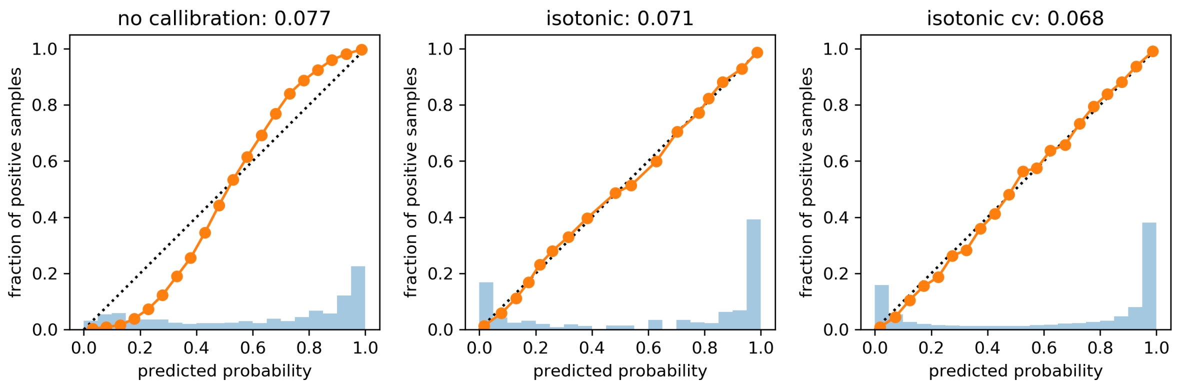


Calibratione con Random Forest

```
cal_rf = CalibratedClassifierCV(rf, cv="prefit", method='sigmoid')
cal_rf.fit(X_val, y_val)
scores_sigm = cal_rf.predict_proba(X_test)[:, 1]

cal_rf_iso = CalibratedClassifierCV(rf, cv="prefit", method='isotonic')
cal_rf_iso.fit(X_val, y_val)
scores_iso = cal_rf_iso.predict_proba(X_test)[:, 1]
```

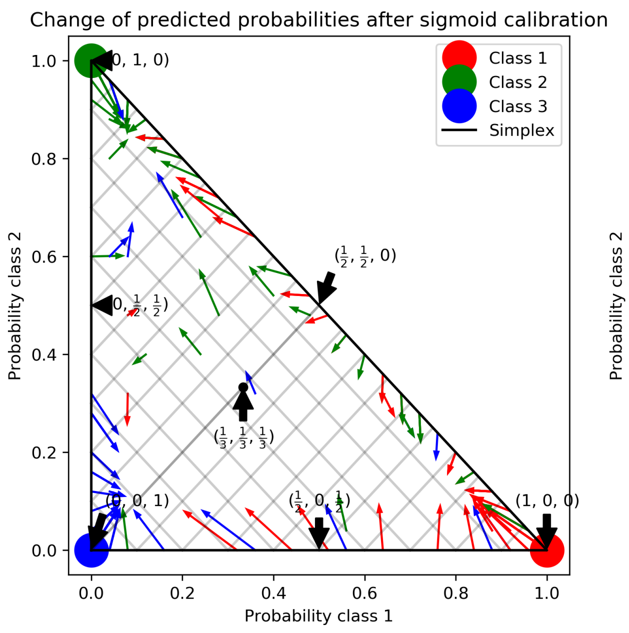


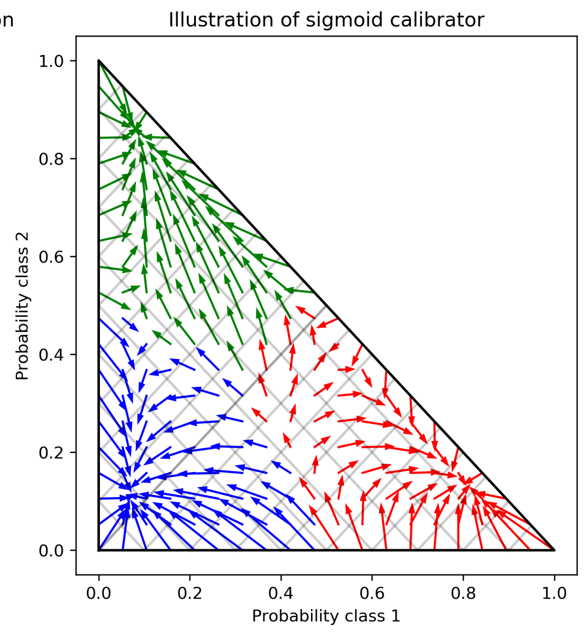
```
cal_rf_iso_cv = CalibratedClassifierCV(rf, method='isotonic')
cal_rf_iso_cv.fit(X_train, y_train)
scores_iso_cv = cal_rf_iso_cv.predict_proba(X_test)[:, 1]
```



Multi-Class Calibration

Ternary Plot -- De Finetti





Riassunto sui dati sbilanciati

Two sources of imbalance

- costo asimmetrico
- dati asimmetrici

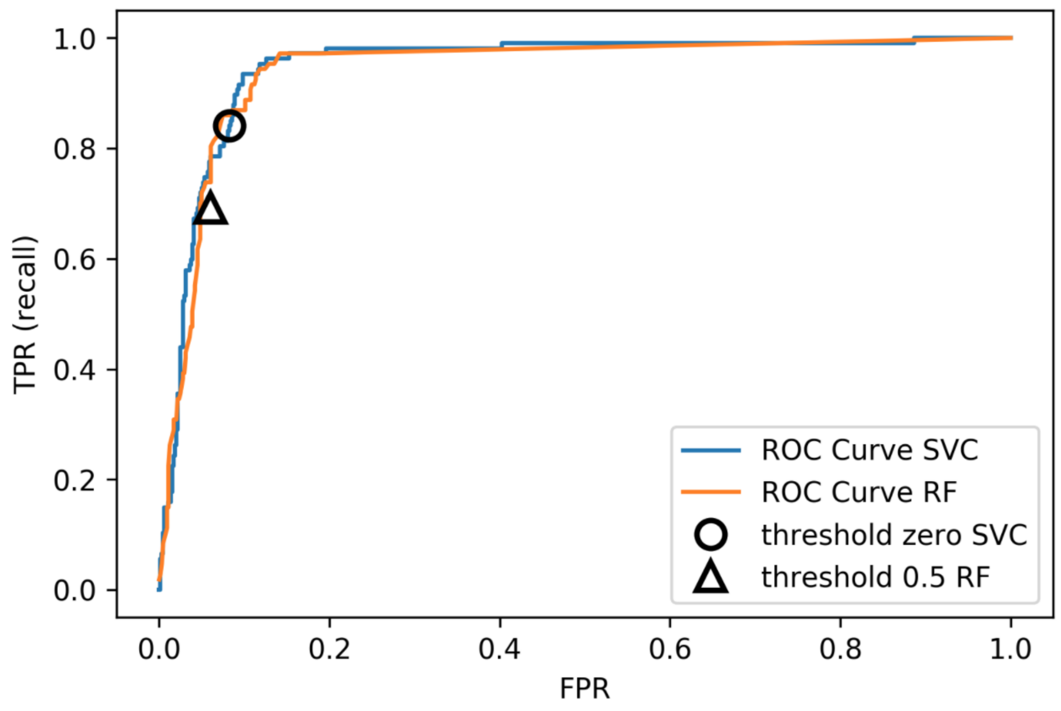
Perché ci importa?

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

Rimedi per il modello

Ricordate? cambiare le Threshold

ROC CURVE



Dati mammografia

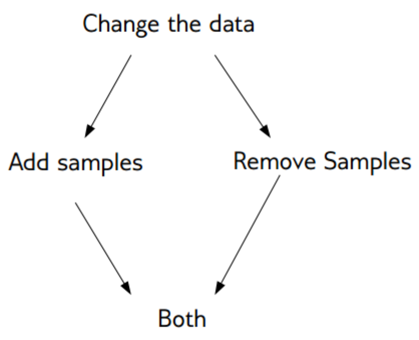
```
from sklearn.model_selection import cross_validate
from sklearn.linear_model import LogisticRegression

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(),
                        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

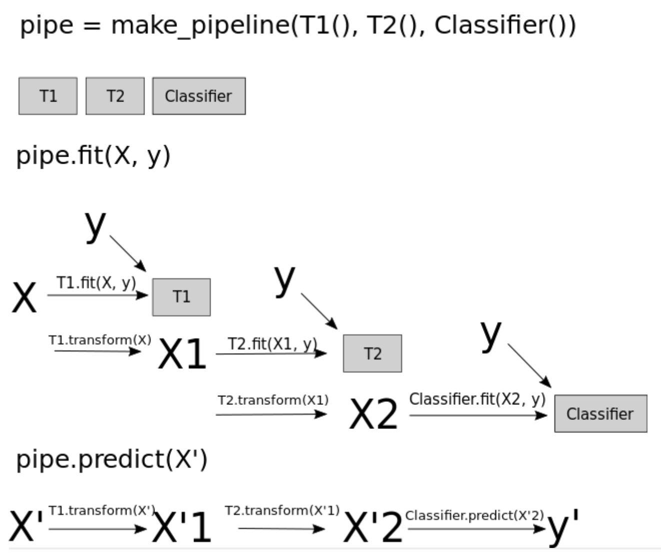
Bilanciamo: schema degli approcci



oppure:

- cambiamo la procedura di addestramento
- o la loss

Scikit-learn vs resampling


pipe = make_pipeline(T1(), T2(), Classifier())
T1
T2
Classifier
pipe.fit(X, y)
y
X
T1.fit(X, y)
T1
y
T1.transform(X)
X1
T2.fit(X1, y)
T2
y
T2.transform(X1)
X2
Classifier.fit(X2, y)
Classifier
pipe.predict(X')
X'
T1.transform(X')
X'1
T2.transform(X'1)
X'2
Classifier.predict(X'2)
y'

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)
```

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] # 195 per classe il 98% e oltre meno sample

```
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 0.920, 0.630
```

0.927, 0.527

- AUC è aumentato ma l'accuratezza è diminuita! Certo abbiamo buttato il 98% dei nostri dati
- Funziona se hai un grande dataset

```
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 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]
```

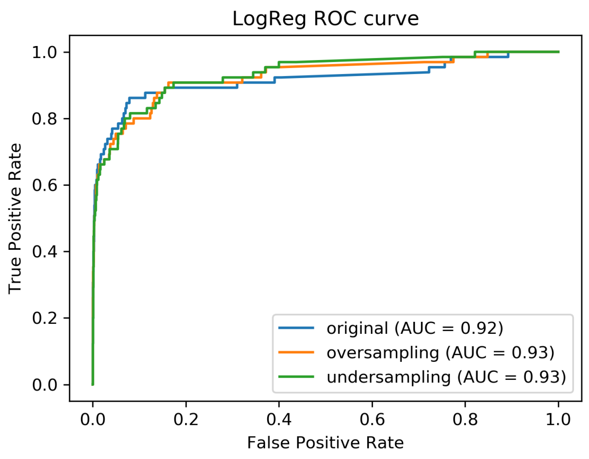
```
oversample_pipe = make_imb_pipeline(RandomOverSampler(), LogisticRegression())
scores = cross_validate(oversample_pipe,
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()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

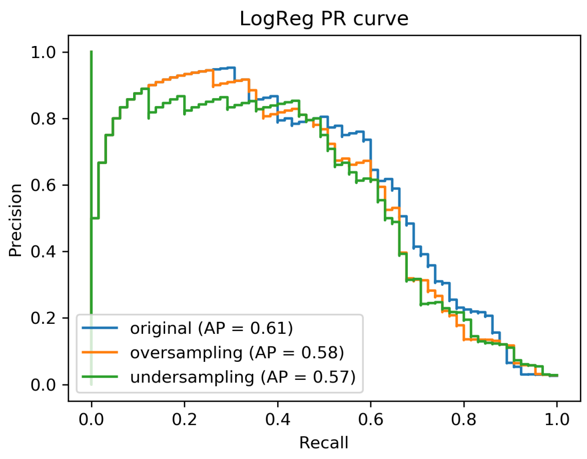
```
0.917, 0.585
```

```
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 0.939, 0.722
```

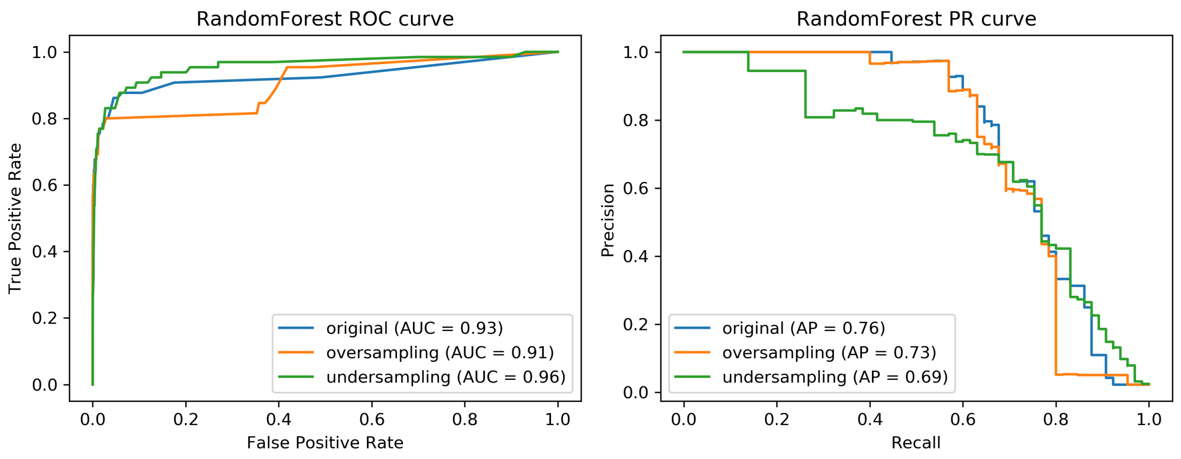
```
0.926, 0.715
```

con la LogReg





con il Random Forest



Pesare le classi

- aggiungo i pesetti $c_{y_i} > 0$

- 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).

peso i modelli lineari

Similar for linear and non-linear SVM

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

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

peso i modelli ad albero

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

usiamo il class_weight 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

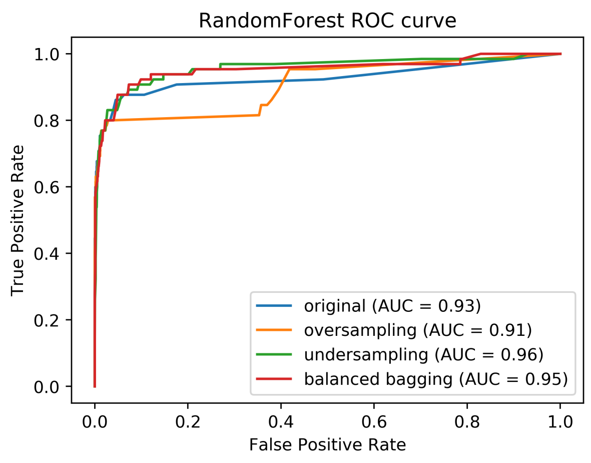
```
scores = cross_validate(RandomForestClassifier(n_estimators=100,
                                               class_weight='balanced'),
                        X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
scores['test_roc_auc'].mean(), scores['test_average_precision'].mean()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

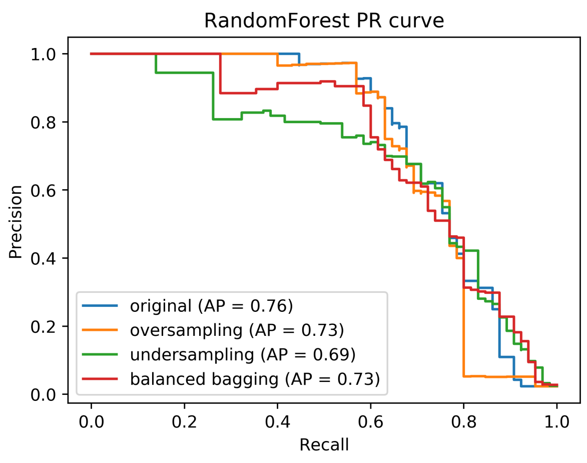
0.917, 0.701

Easy Ensemble in 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,
                                         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

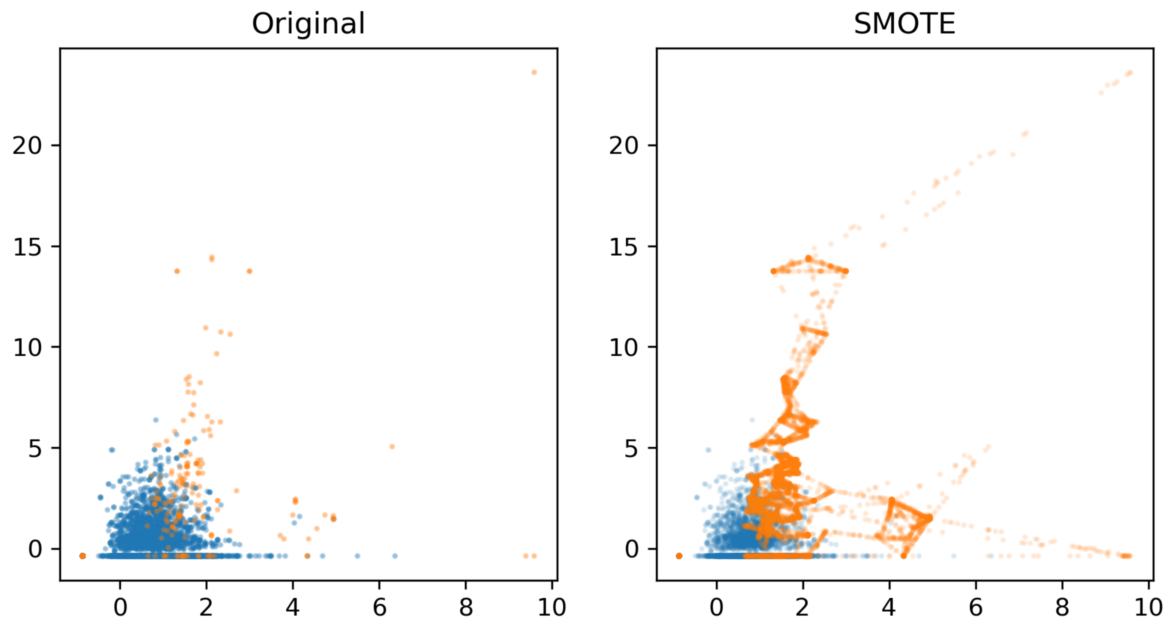




Generazione di Sample Sintetici

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.



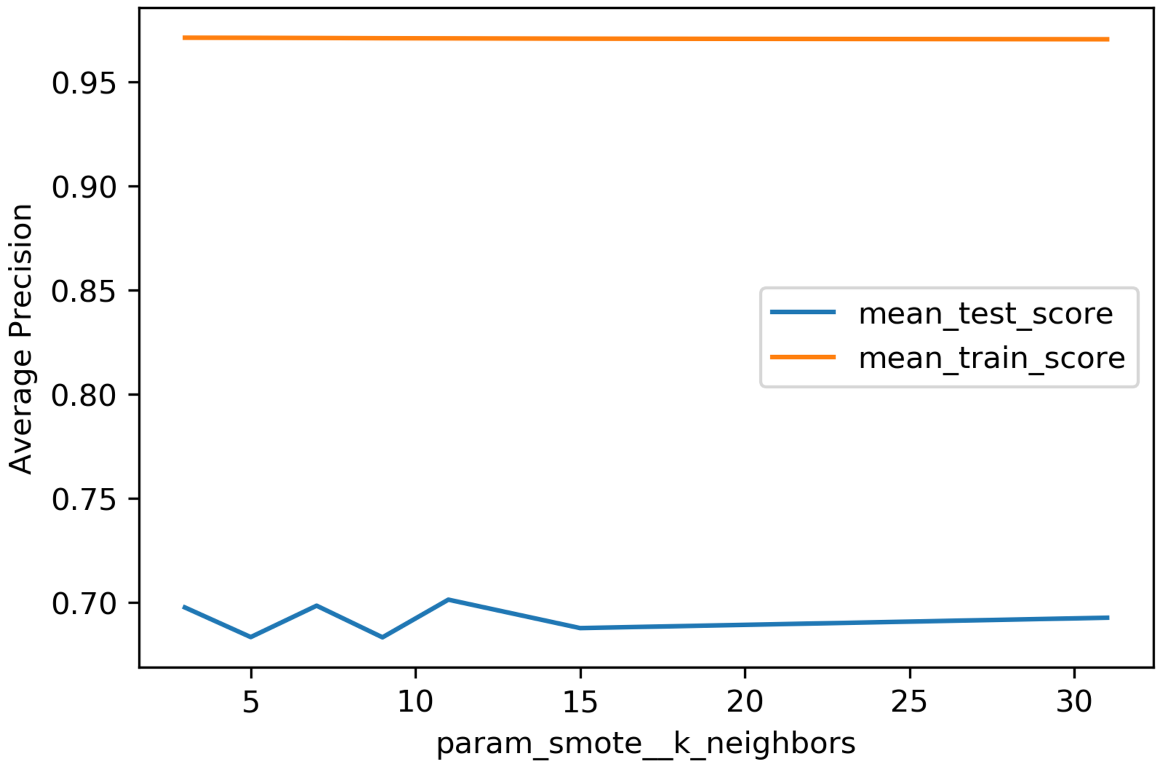
```
smote_pipe = make_imb_pipeline(SMOTE(), LogisticRegression())
scores = cross_validate(smote_pipe, X_train, y_train, cv=10,
                        scoring=('roc_auc', 'average_precision'))
pd.DataFrame(scores)[['test_roc_auc', 'test_average_precision']].mean()
# baseline was 0.920, 0.630
```

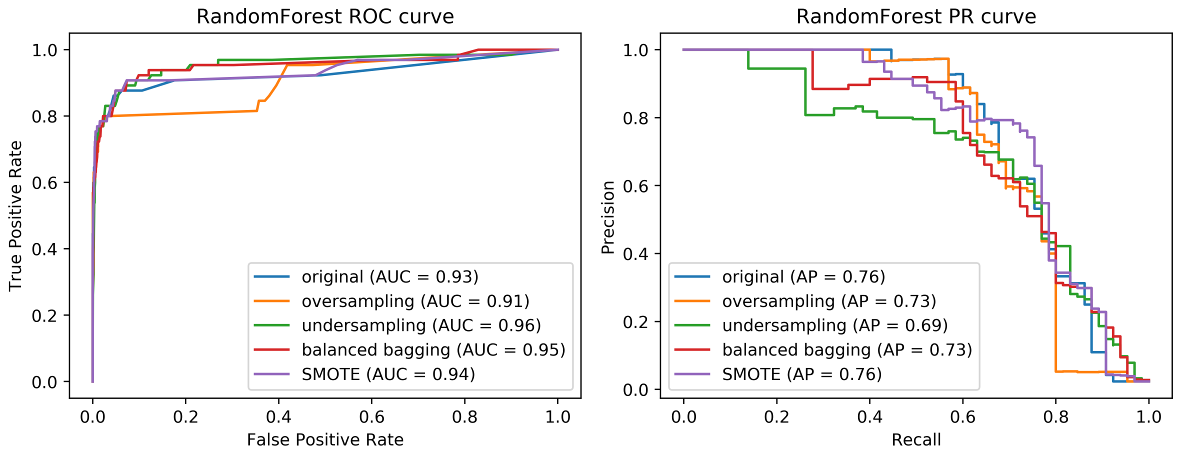
0.919, 0.585

```
smote_pipe_rf = make_imb_pipeline(SMOTE(),
                                  RandomForestClassifier())
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"]
```





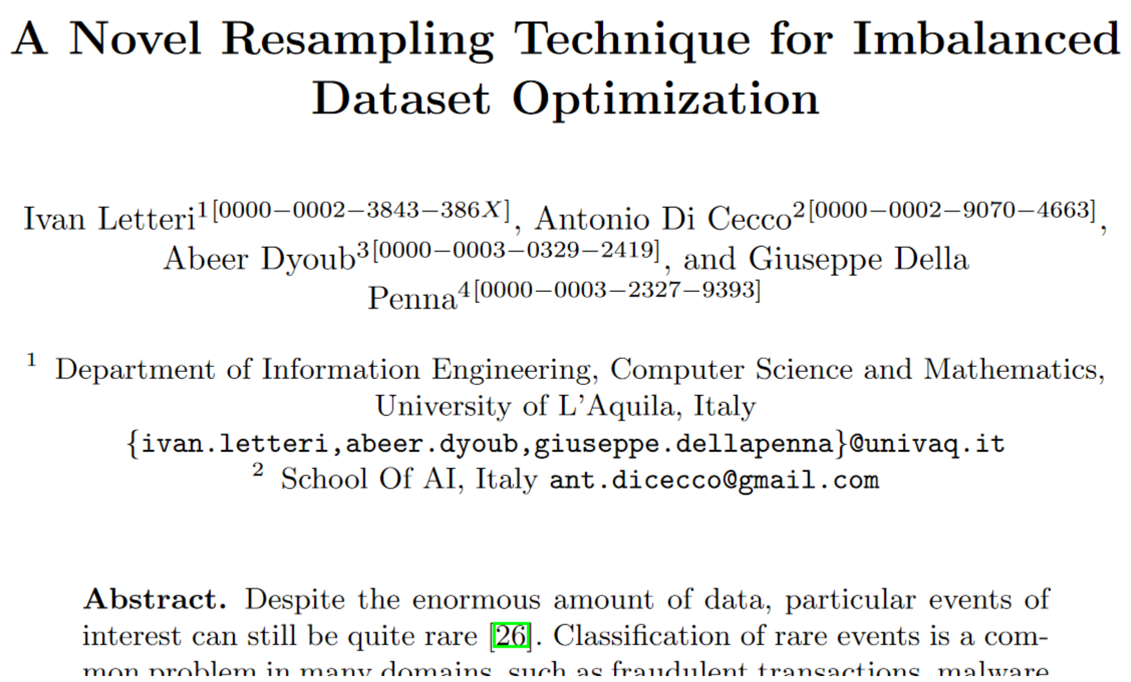
I DIFETTI PRINCIPALI DI SMOTE

- non estrapola
- non segue la corretta distribuzione di densità
- aumenta la collinearità nel dataset

Soluzioni più moderne sono generare nuovi sample usando metodi generativi come le GAN ma le GAN sono notoriamente instabili per questo difficili da addestrare

Una soluzione ‘G1no’ che sta per Gaussian o Generative1NN.

- G1no is ALL YOU Need



A Novel Resampling Technique for Imbalanced Dataset Optimization

Ivan Letteri[1][0000−0002−3843−386X], Antonio Di Cecco[2][0000−0002−9070−4663], Abeer Dyoub[3][0000−0003−0329−2419], and Giuseppe Della Penna[4][0000−0003−2327−9393]

[1] Department of Information Engineering, Computer Science and Mathematics, University of L’Aquila, Italy
{ivan.letteri,abeer.dyoub,giuseppe.dellapenna}@univaq.it
[2] School Of AI, Italy ant.dicecco@gmail.com

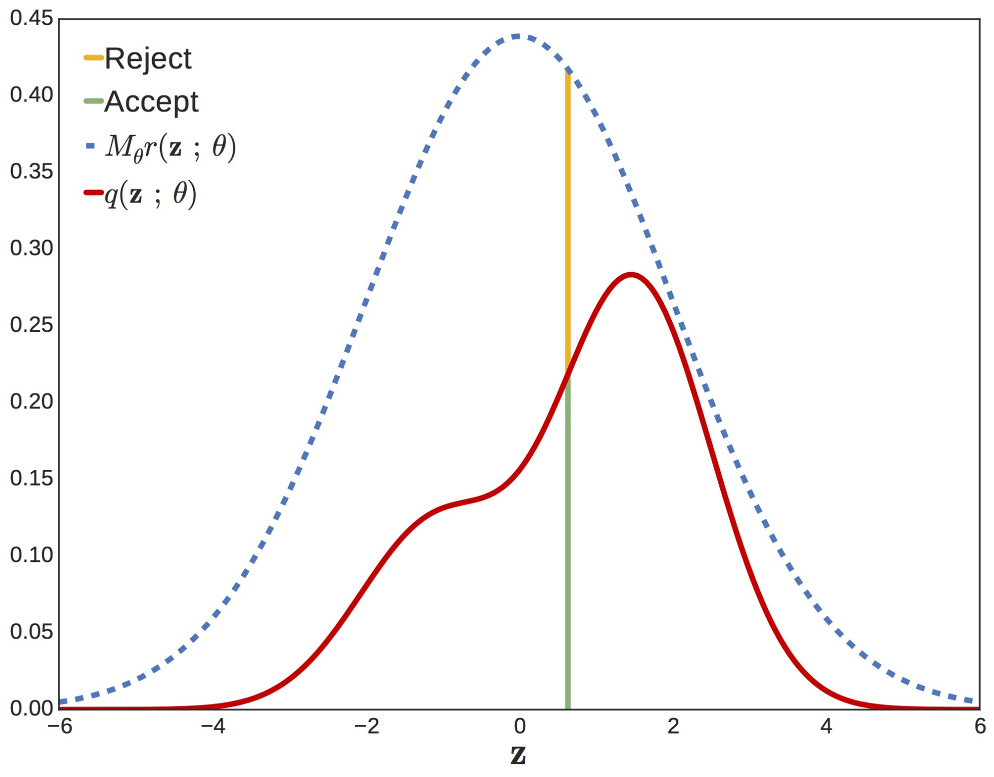
Abstract. Despite the enormous amount of data, particular events of interest can still be quite rare [26]. Classification of rare events is a common problem in many domains, such as fraudulent transactions, malware

https://arxiv-export-lb.library.cornell.edu/pdf/2012.15231

A nostra conoscenza nuovo stato dell’arte per quanto riguarda l’oversampling dei dati tabulari G1no sarà presto in imblearn.

Cosa fa G1no?
Usa un modello non sensibile all’unbalance per generare nuovi campioni questo modello è il Nearest Neighbours e lo trasformo in un metodo generativo.
Come si trasforma un modello in un modello generativo?

Si usa il rejection sampling (di Von Neumann): estraggo rispetto a una distribuzione di probabilità controllo se il NN è d’accordo con la classificazione se si lo aggiungo nella classe di minoranza.



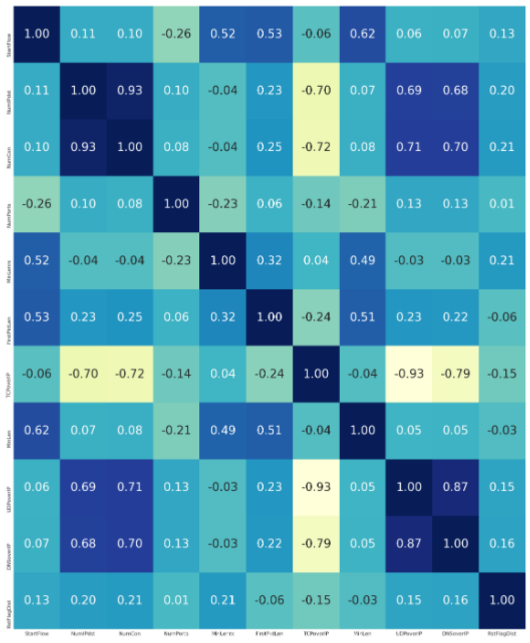
Il sampling è una semplice distribuzione gaussiana su tutti i sample dell'insieme di train.
Considero le features indipendenti ecc ecc.

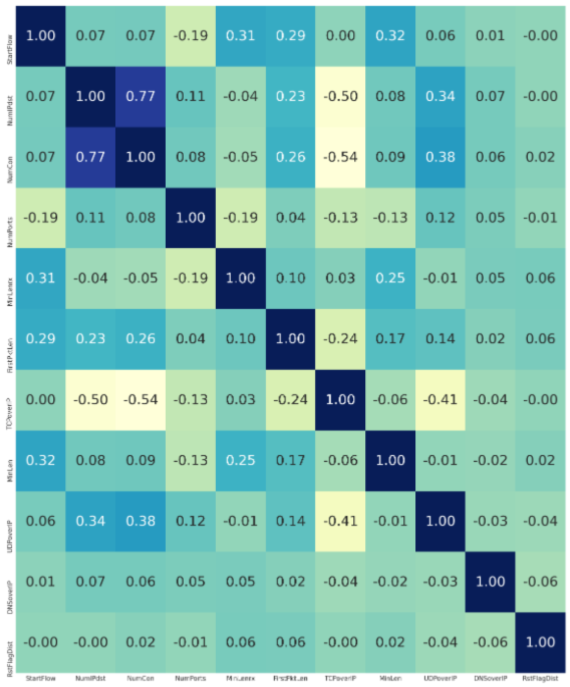
G1no è una vera tecnica di *augmentation* una specie di coltellino svizzero...
Ha anche un meccanismo che migliora ogni modello.

Nella generazione gaussiana ho come 'prior' che le features sono indipendenti e quindi il campione generato conserva questa proprietà.

Modelli lineari o quasi lineari (NN, SVM) sono facilitati nell'apprendimento quando le features sono indipendenti (e non collineari).

Anche i sistemi ad albero funzionano meglio. Perché un sistema ad albero considera più importanti features continue che categoriche (la ricerca degli split). Sfumare le features categoriche rende la ricerca degli split più efficace.


(a) Correlation Matrix of SMOTE


(a) Correlation Matrix of G1No

	Precision	**F1**	**AUC**	**Accuracy**
SMOTE	99.52	99.50	99.90	99.49
ADASYN	99.80	99.19	99.80	99.20
G1No	**99.82**	**99.69**	**100.0**	99.69
G1No Gourmet	**99.84**	**99.67**	**100.0**	99.70

FINE