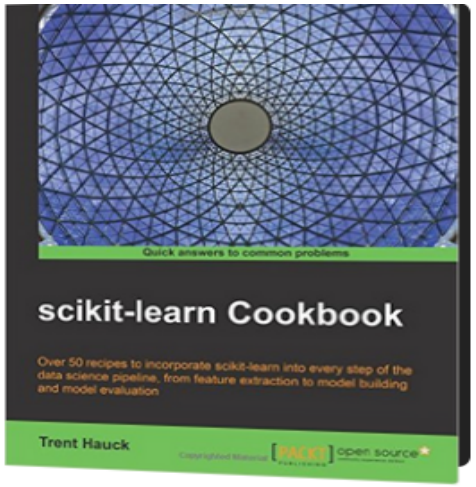
Austin ACM SIGKDD - Austin's Big Data Machine Learning Group

Advanced Machine Learning *with* Python



Sponsored by VISA

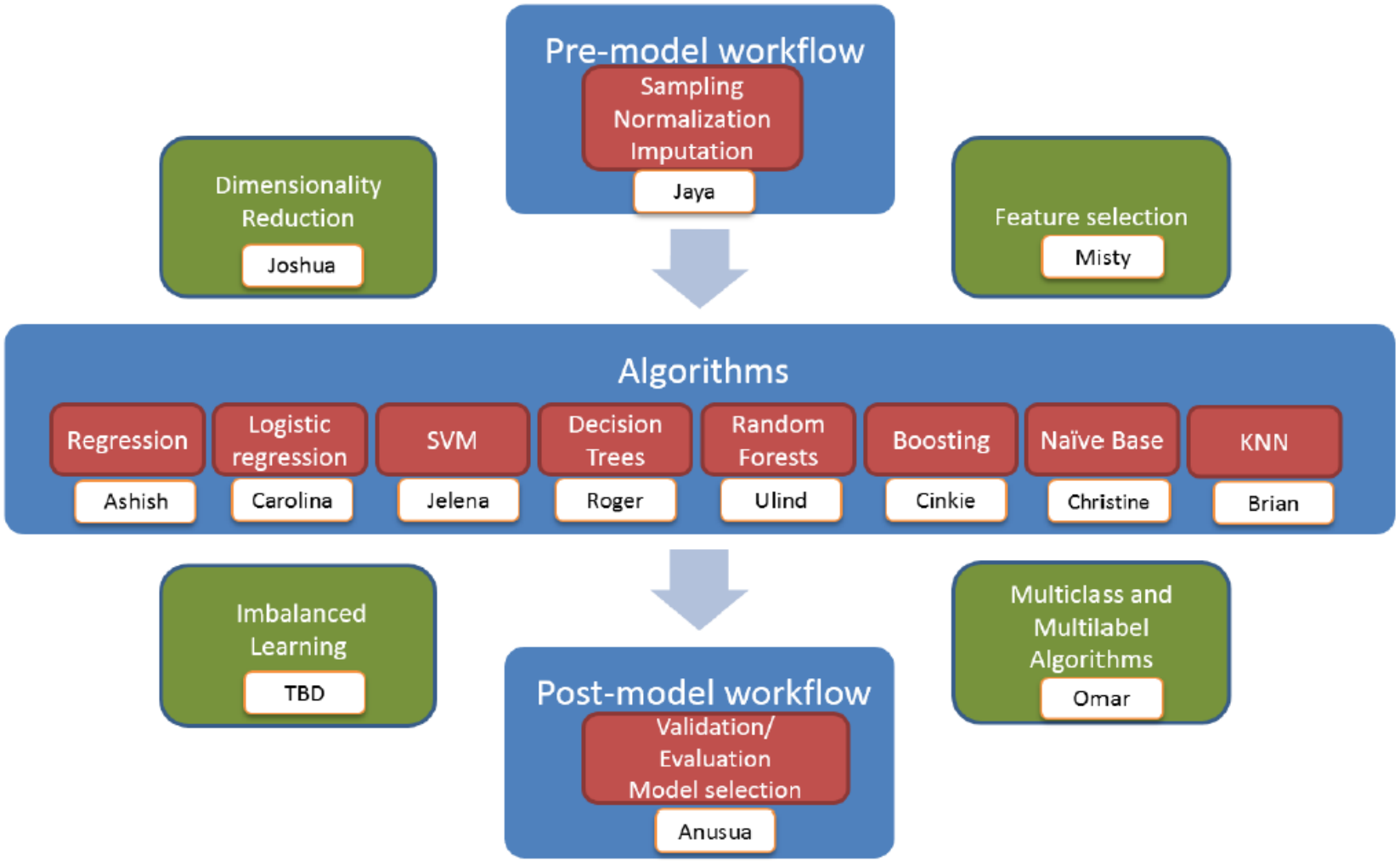
WiFi connection

- Connect to “other network”.
- Type in the network name. It is **lightning**
  - all lower case letters.
  - NO e in the middle
- Mac users select WPA2 Personal.
- Then type the password **zeuswgafc**

KDD bringing together the data mining, data science and analytics community

Austin ACM SIGKDD - Austin's Big Data Machine Learning Group

Advanced Machine Learning *with* Python



Advanced Machine Learning with Python

Session 7: Imbalanced Learning

Tim Gasser

github.com/timgasser/acm_imbalanced_learning

@tim_gasser

medium.com/@timgasser

Overview

- Introduction to imbalanced learning
- Evaluation metrics
- Imbalanced learning approaches
  - Sampling
  - Algorithms
- Kaggle dataset evaluation
- Conclusions

Introduction to imbalanced learning

- Formal definition of imbalanced learning [1]: '…the learning process for data representation and information extraction with severe data distribution skews to develop effective decision boundaries to support the decision-making process.'
- Informally, imbalanced learning is the training of machine learning algorithms on datasets where far fewer examples exist of one class than others.
- The imbalance ratio is the ratio between classes of each type. It can vary between 100:1 to 10,000:1.
- This presentation will focus on imbalanced learning with classification algorithms.

Introduction to imbalanced learning (contd)

- Classification use-cases with imbalanced data
  - Fraud detection, anomaly detection, cancer detection, predicting credit defaults
- In imbalanced learning, prediction of the minority class examples is more important than the majority class.
  - Unfortunately the algorithm has far of these fewer minority class examples to learn from.
- Standard algorithms focus on reducing the cost of misclassifying all examples, not just minority ones.
- In future slides, I'm assuming we're predicting the minority class (so positive means minority example).

Evaluation Metrics

Evaluation of imbalanced learning

- When training classifiers, accuracy is often used to quantify how good the model is.
- Accuracy is defined as TP + TN / N
- Take a dataset with a 95:5 imbalance ratio
  - The most naïve model predicts the most common case for every example.
  - This has 95% accuracy on the dataset above !
  - But this mis-classifies **all** the minority class examples.
- We need a different metric to optimize!

Alternative imbalanced metrics

- Precision (TP / TP + FP) and Recall (TP / TP + FN) give more useful information about the minority-class performance of the model. But using two figures as a metric is inconvenient.
- Alternative is to use the F1 score (harmonic mean of precision and recall) = 2TP / (2TP + FP + FN)
- A more visual representation of the model's performance is the Receiver Operating Characteristic (ROC).
  - The Area Under the Curve (AUC) is a single metric to quantify model performance.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Area under Roc Curve

- The classification algorithm outputs a probability of the example being a positive case (between 0 and 1).
- This probability is converted into a positive/negative by applying a decision threshold.
- The ROC Curve shows how the FPR and TPR vary as this threshold is varied from 0 (top right) to 1 (bottom left).
- The area under the curve (AUC) is independent of the threshold, and shows how well the model is performing.

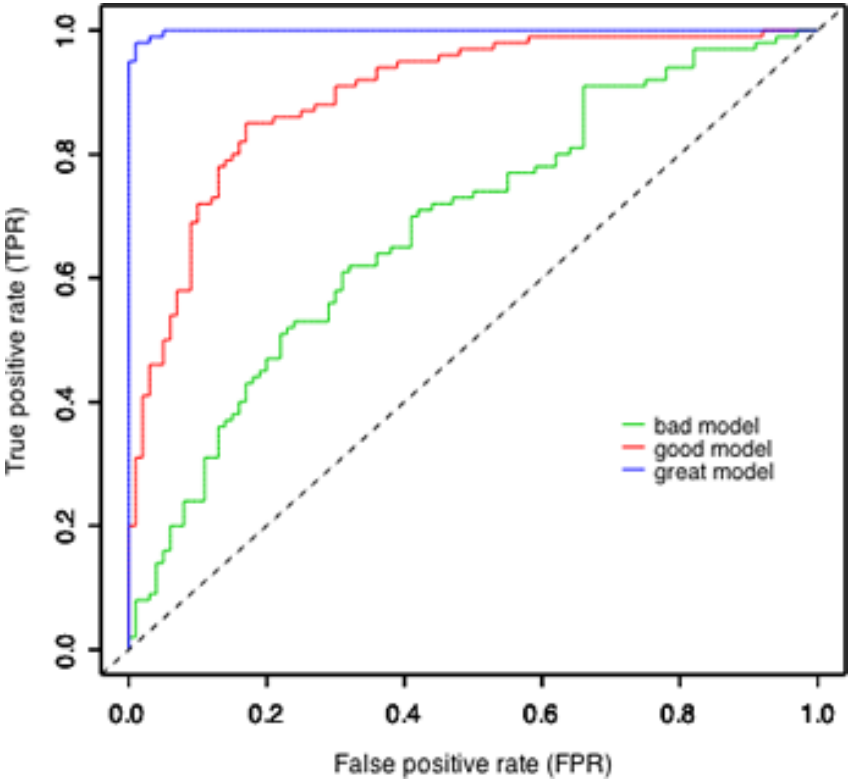


Image source[3]

ROC Curves and decision thresholds

- The decision threshold can be adjusted based on the relative costs of FPs and TPs.
- For example in cancer detection:
  - False Positive: The patient has a further un-necessary scan to confirm the incorrect diagnosis.
  - True Positive: Cancer is correctly diagnosed.
- There is a good case for decreasing the threshold of the model.
  - This will increase the TPR, with a small increase in FPR.

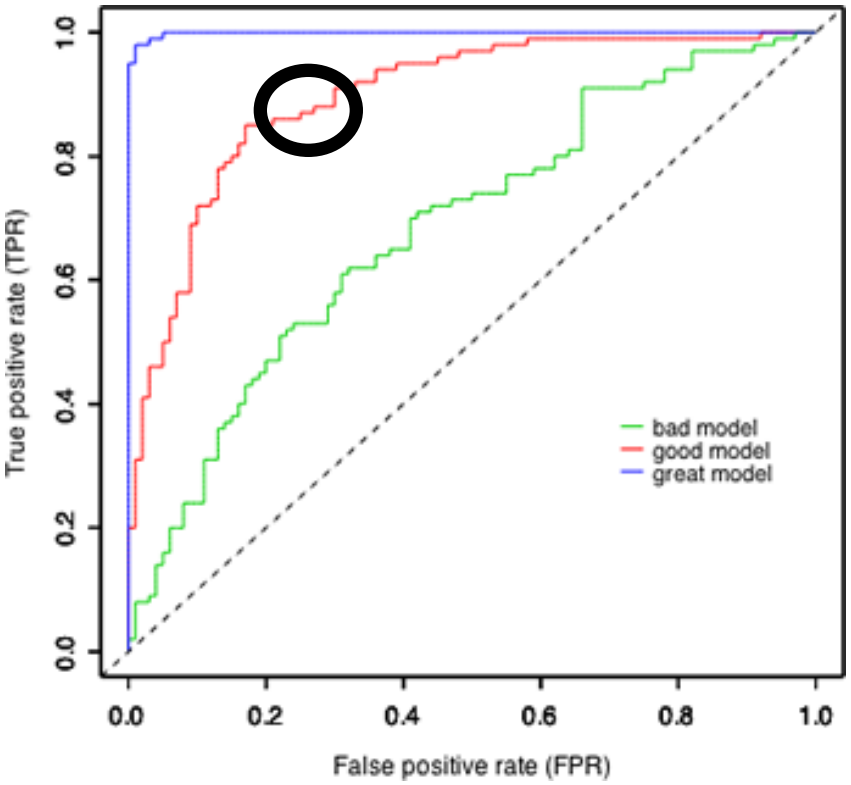
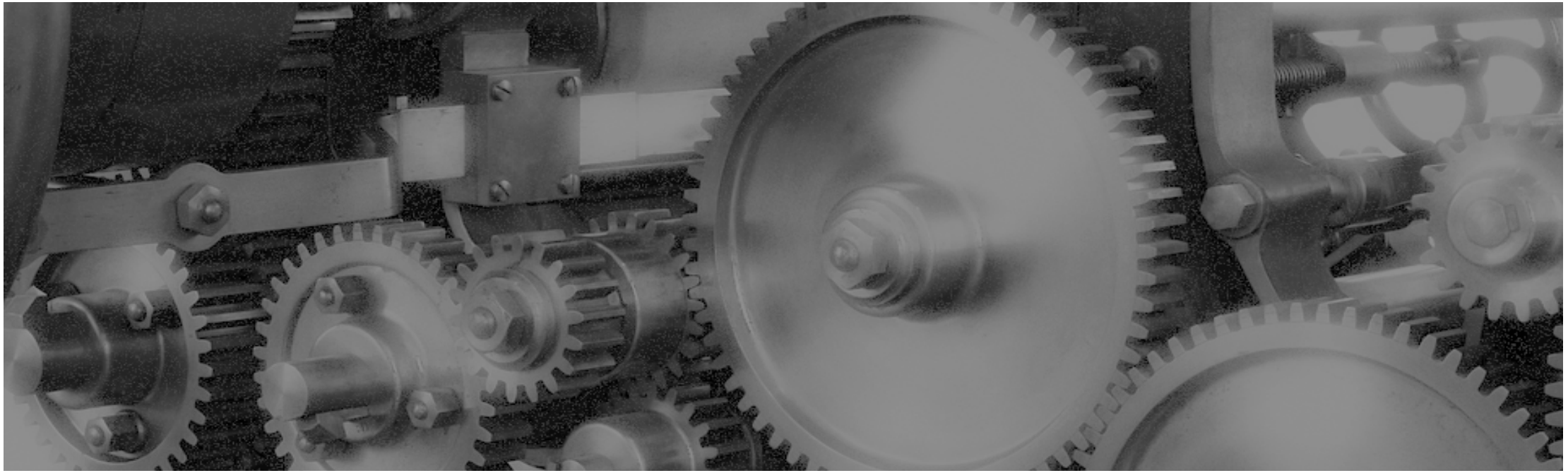


Image source[3]



Imbalanced learning techniques

Imbalanced learning techniques

- To get good predictive performance on imbalanced datasets, you can do the following
  - Data pre-processing. This modifies the dataset to reduce the imbalance before training the model.
  - Algorithm-specific. Depending on the algorithm used, weighting can be used to increase the penalty for mis-classifying a minority example.
  - Ensemble. By training multiple models and combining the results, performance can be increased over a single model. This is not just true for imbalanced datasets.

Data pre-processing

Data pre-processing

- There are two options to generate a 50:50 balanced dataset from an imbalanced one
  - Oversampling: Generate new minority class examples.
  - Undersampling: Remove majority class examples.
- These can be done at random, but there are issues
  - Random oversampling gives issues with overfitting, as there will be many identical duplicated minority examples.
  - Random undersampling means you throw away good majority data.
- To improve on the performance of random under and oversampling, kNN-based methods selectively add or remove data.

Oversampling methods

- Synthetic Minority Oversampling Technique (SMOTE)
- For each minority example
  - Randomly chose one of the k nearest neighbours (majority or minority)
  - Create a new sample a random distance between the minority example and the nearest neighbour.
  - This allows the minority example class space to be expanded.

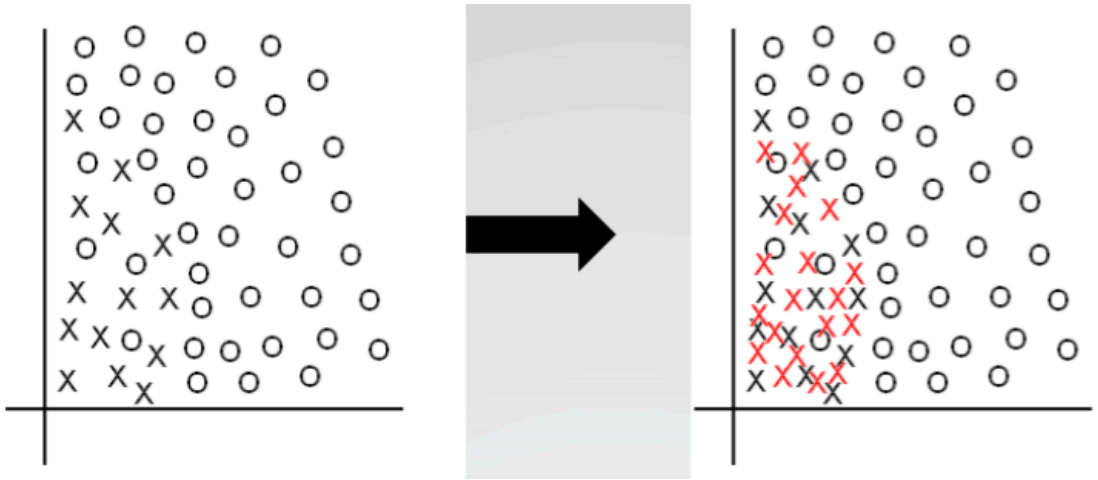


Image source[2]

Undersampling methods

- Tomek Links
  - Find pairs of points that are different classes, they form a Tomek link if there is no closer example to either point.
  - Remove majority example in the pair.
  - Effectively widens the decision boundary between majority and minority.

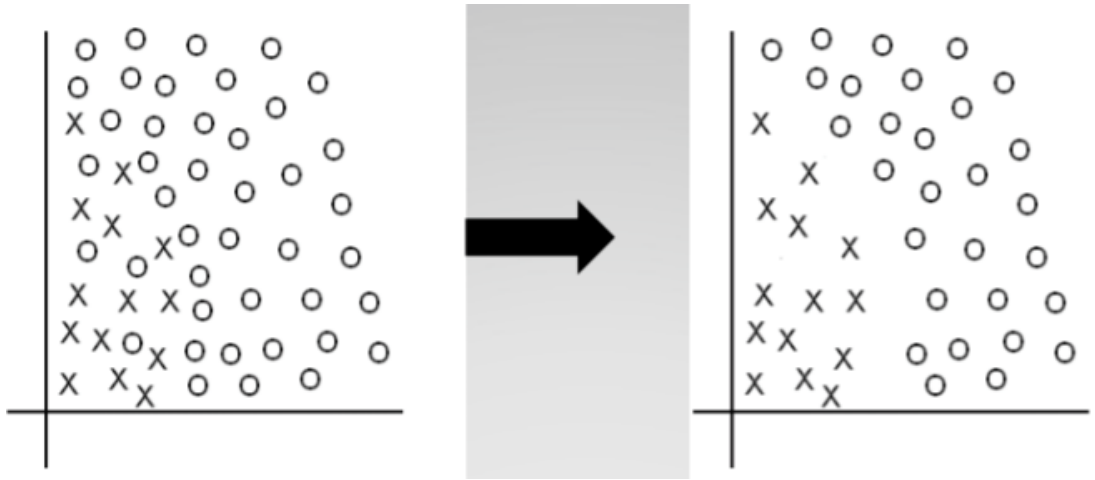


Image source[2]

Undersampling methods

- Condensed Nearest Neighbor (CNN)
  - Removes all points, and adds them back in as required to correctly predict the examples with a kNN classification where k=1.
  - CNN removes majority class examples that are distant from the decision border.

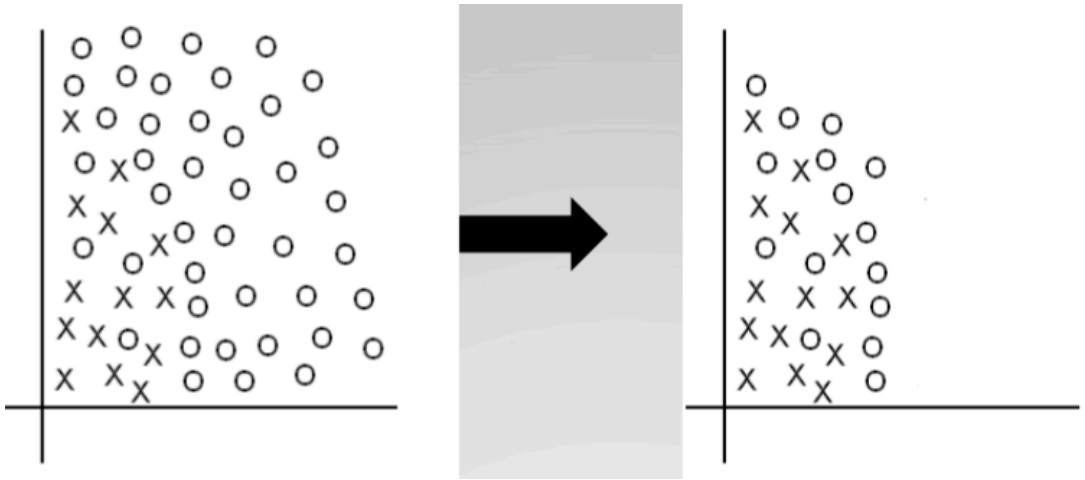


Image source[2]

Combining undersampling methods

- One-Sided Selection (Tomek links followed by Condensed Nearest Neighbor Rule)
  - Removing majority-class examples from Tomek links increases the separation between majority and minority class feature spaces.
  - Condensed Nearest Neighbor Rule removes majority examples far from the class borderline.

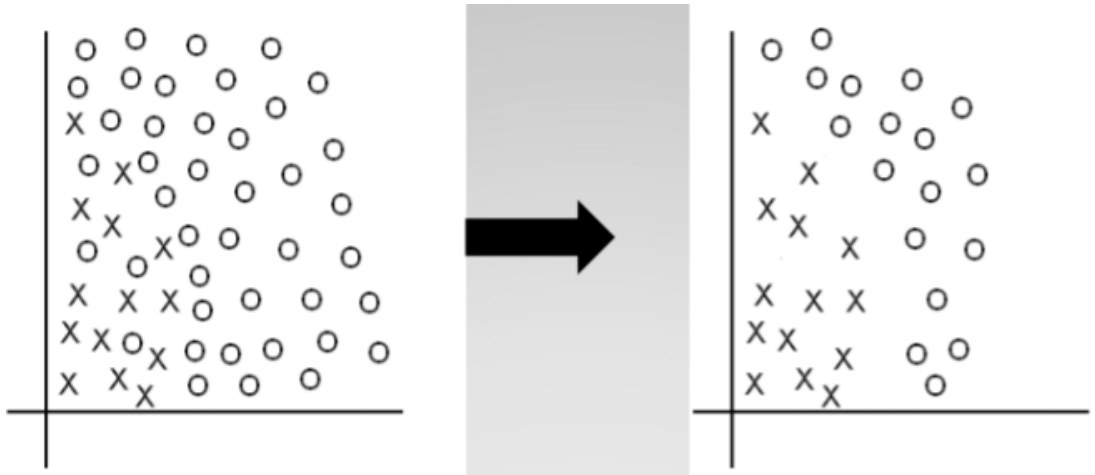


Image source[2]

Data pre-processing tools and APIs

- Python's scikit-learn library doesn't support any of the dataset processing methods. These don't fit well with the sklearn API:
  - https://github.com/scikit-learn/scikit-learn/issues/4617
- I found a repo on Github which supports all the techniques described and more besides:
  - https://github.com/fmfn/UnbalancedDataset
- R has a package called 'unbalanced' which also supports all the pre-pre-processing modes:
  - https://cran.r-project.org/web/packages/unbalanced/unbalanced.pdf
- To understand the algorithms, I coded several using a small example in the the notebook. Demo !
  - acm_imbalance_sampling.ipynb

Cost-sensitive Algorithms

Solutions – Cost-sensitive methods

- With a dataset of 95:5 imbalance, mis-classifying the minority examples can contribute a maximum of 5% of the loss.
- This will skew the model towards correctly classifying the majority samples !
- To solve this, we can weight the classifications using the inverse of class frequency.
- So scale each mis-classification of the minority class by the imbalance
  - Scaling the minority class cost by 95/5 = 19 gives a cost function ratio of 95:95.

Solutions – Cost-sensitive methods

- The following classifiers in scikit-learn can adjust weights inversely to class frequencies
  - sklearn.linear_model.LogisticRegression
  - sklearn.tree.DecisionTreeClassifier
  - sklearn.svm.LinearSVC
- The API is standard across these classifiers (copied below)
  - `class_weight : dict or 'balanced', optional`
  - `The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as n_samples / (n_classes * np.bincount(y))`
- To test the performance of the weighted classifiers, I'm going to use a real-world dataset.

Real-world use-case

Evaluation dataset

- For this presentation, we'll use the Kaggle 'Give Me Some Credit' dataset from Dec 2011
  - https://www.kaggle.com/c/GiveMeSomeCredit .
- The aim of the competition was to 'Improve on the state of the art in credit scoring by predicting the probability that somebody will experience financial distress in the next two years.'
- Evaluation metric is 'AUC' - Area Under (ROC) Curve.

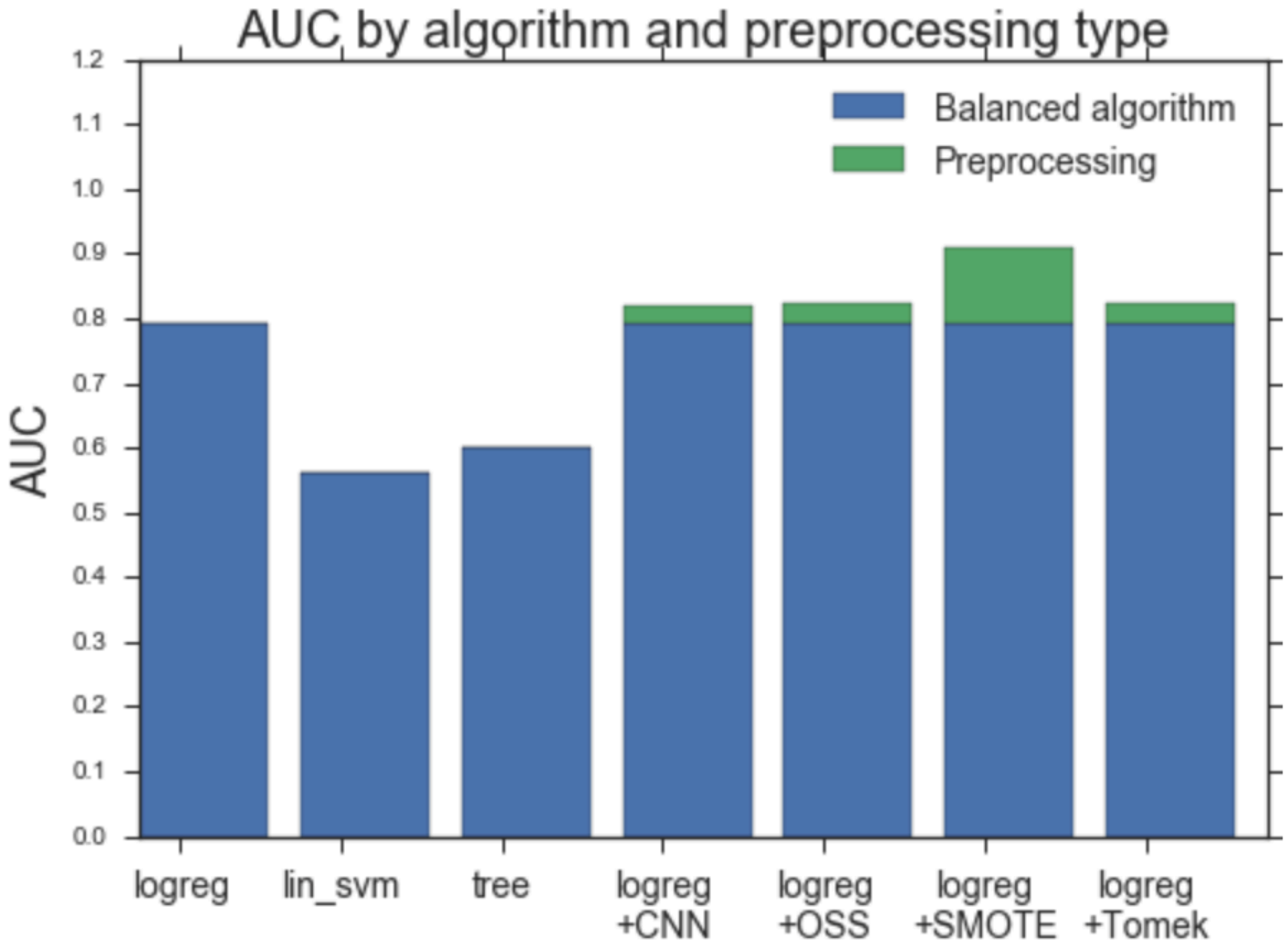
Evaluation dataset (features)

Variable Name	Description	Type
SeriousDlqin2yrs	**Person experienced 90 days past due delinquency or worse**	**Y/N**
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits	percentage
age	Age of borrower in years	integer
NumberOfTime30-59DaysPastDueNotWorse	Number of times borrower has been 30-59 days past due but no worse in the last 2 years.	integer
DebtRatio	Monthly debt payments, alimony,living costs divided by monthy gross income	percentage
MonthlyIncome	Monthly income	real
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards)	integer
NumberOfTimes90DaysLate	Number of times borrower has been 90 days or more past due.	integer
NumberRealEstateLoansOrLines	Number of mortgage and real estate loans including home equity lines of credit	integer
NumberOfTime60-89DaysPastDueNotWorse	Number of times borrower has been 60-89 days past due but no worse in the last 2 years.	integer
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)	integer

Evaluation overview

- The first section of the notebook explores the dataset, and compares classifiers trained with accuracy instead of AUC, and with class weights.
- The second section uses pre-processed datasets generated in R to evaluate the performance boost.
- “Which is better, R or Python?”
  - Use best libraries from both ☺

Evaluation results



Evaluation results

- Logistic regression (using weighted classes) in combination with SMOTE is the best performing model, with AUC = 0.909667.
- Evaluation on the Kaggle ‘Give me some credit’ dataset shows:
  - Training with the correct metric is crucial.
  - Balancing class weights is important for a 95:5 imbalanced dataset.
  - Pre-processing the data with SMOTE gives a significant boost in AUC.

Conclusions

- scikit-learn does not support any data pre-processing, but does support weighted classes in classifier algorithms.
- The ’unbalanced’ R package does support data pre-processing.
  - But using R in the pipeline complicates the flow slightly.
- Ensemble algorithms are another solution for class imbalance, unfortunately there wasn’t time to present them here.
  - See BalanceCascade and EasyEnsemble (in [4]), for more information.

Thank you !

Any questions?

References

- [1] - Imbalanced Learning: Foundations, Algorithms, and Applications (1st ed), Haibo He, Yunqian Ma (link).
- [2] – Mining when classes are imbalanced, rare events matter more, and errors have cost attached, **SIAM** Data Mining Conference (SDM **2009**), Nitesh V. Chawla (link).
- [3] - ECOL/BIOL 563 Statistical Methods in Ecology, Fall 2010 (link)
- [4] - "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 39, no. 2, pp. 539–550, 2009. X.-Y. Liu, J. Wu, and Z.-H. Zhou.
- [5] - K. Chidananda and G. Krishna, "The condensed nearest neighbor rule using the concept of mutual nearest neighbor", IEEE Trans. Information Theory, Vol IT- 25 pp. 488-490, 1979.