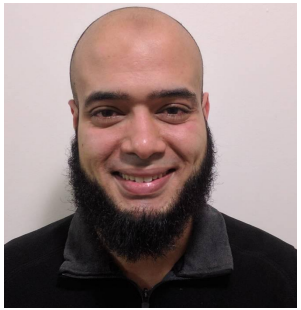
Machine Learning

Imbalanced Datasets

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)



© 2023 All rights reserved.
Please do not reproduce or redistribute this work without permission from the author

Recall: Imbalanced dataset

- When one or more of the classes has too many labels and while some are very little examples, we call it imbalanced dataset
- Credit card transactions: 99.9% of legitimate transactions (negative) and only 0.1% of fraud (positive)
  - In 1000,000 examples, only 1000 are fraud
- If the classifier decided just predict EVERYthing as legitimate, we get accuracy of 99.9%, but this is very **biased** classifier (toward -ve)
- **Tip**: In imbalanced datasets, both **accuracy and log-loss** might be used as indicator if the model is wrong, but shouldn't be used for the opposite
  - 55% accuracy ⇒ bad mode. 99% accuracy ⇒ probably majority focused model

Consider

- In these slides, I call the **minority** class the **positive** class
- In these slides, we assume our main concern is **positive** class
- Let P be the total number of positive samples (minority)
- Let N be the total number of negative samples (majority)

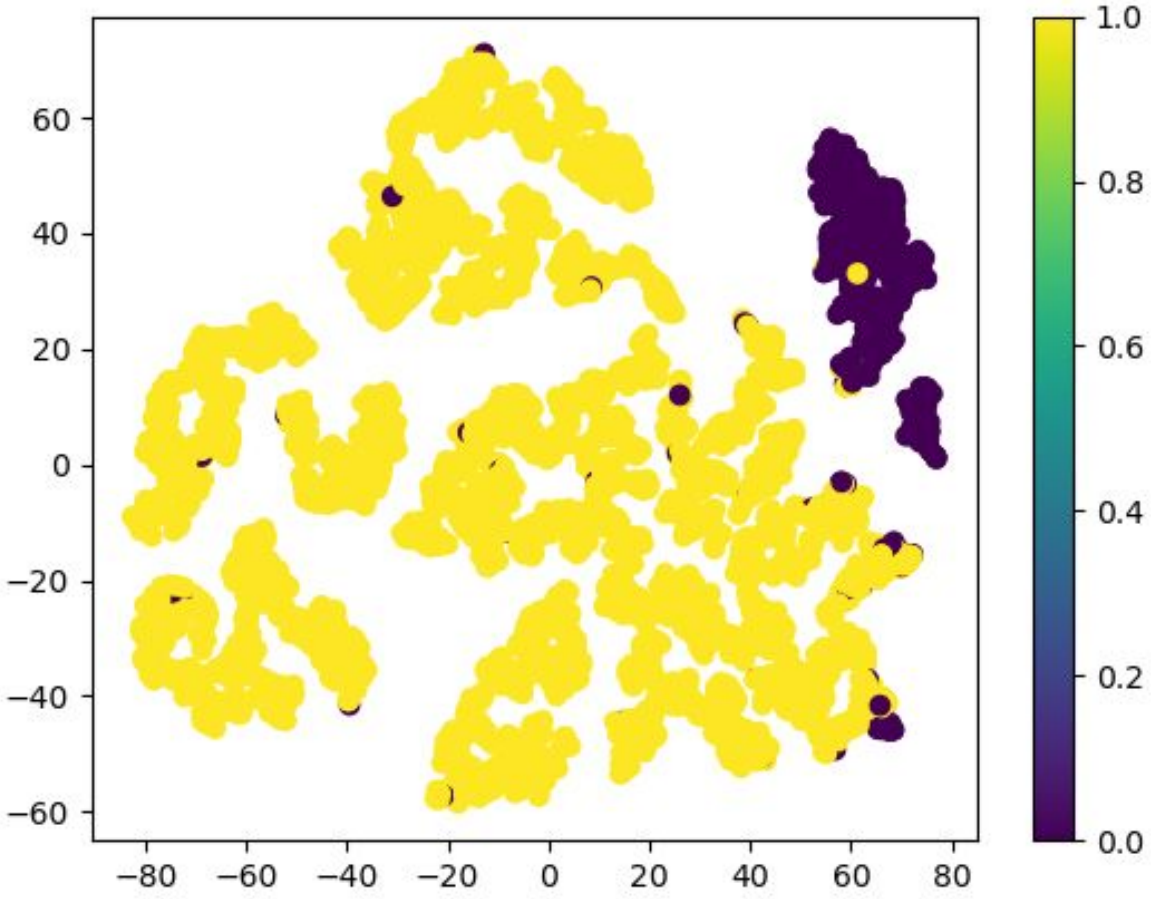
Let’s visualize: sample negative class

```
def visualize(X, y):
    # Project to 2 features
    # https://www.geeksforgeeks.org/difference-between-pca-vs-t-sne/
    tsne = TSNE(n_components=2)
    X_embeded = tsne.fit_transform(X)

    plt.scatter(X_embeded[:, 0], X_embeded[:, 1], c=y)
    plt.colorbar()
    plt.show()

if __name__ == '__main__':
    X, y = make_classification(n_samples=5000, n_features=5,
                               n_informative=2, n_redundant=3,
                               n_clusters_per_class=1, weights=[0.1])

    visualize(X, y)
```


60
40
20
0
−20
−40
−60
−80
−60
−40
−20
0
20
40
60
80
1.0
0.8
0.6
0.4
0.2
0.0

Dealing with imbalanced datasets

- Imbalanced datasets cause bias towards the majority class
- There are some classical approaches to handle that
- However, first consider the following:
  - Adding **more positive examples** can be more successful approach
  - A missing advise is: try to build a good model using **the data as it is**
    - The imbalance reflects the actual distribution rather than a problem to solve
    - Sometimes, the coming techniques (add/remove) just does nothing (or even harm)
      - You may see this we NN and DNN (redundancy may not help)
      - In computer vision, augmented data is an added value (compare to tabular)
  - Classical solutions may not work
    - You may need to be creative and find ad-hoc steps to tackle the problem

Strategies to handle imbalanced datasets

- Undersampling majority class
- Oversampling minority class
  - Random **duplicates**
  - **Synthetic** Data: SMOTE
  - Data **Augmentation**
- Undersampling majority + Oversampling minority
- Cost-Sensitive Training
- Anomaly Detection approach
- Tree-based Algorithms

Undersampling

- Simply we remove some elements from the majority class!
- Cons: can discard potentially useful data
- In practice, it is still a good trial to ‘understand’ the effect of the majority on the model’s bias and think about it
  - It can also give you initial good baseline to deploy while working on better one
- Assume the majority class is 100k example and the minority is 1000
  - We can sample factor x 1000 from the majority
  - Factor can be a small number, e.g. {1, 2, 3, 4}
    - Why? Factors > 1 may allow some useful bias toward the majority class

Undersampling

- imbalanced-learn (imblearn) provides techniques for imbalance data
  - *pip install scikit-learn==1.0.2*
  - *pip install imblearn (imb for imbalanced) - neeed specific scikit version?*
- **RandomUnderSampler**: randomly selecting a subset of data
- There is also NearMiss Algorithms

```
counter = Counter(y)
print(counter)  # Counter({1: 4923, 0: 77})

from imblearn.under_sampling import RandomUnderSampler
factor, minority_size = 2, counter[0]   # 77
rus = RandomUnderSampler(sampling_strategy={1: factor * minority_size},
X_us, y_us = rus.fit_resample(X, y)
print(Counter(y_us))                    # Counter({1: 154, 0: 77})
```

Oversampling

- Over-sampling creates **new instances** from the minority class.
- This can be done by in 2 ways
- 1) Random over-sampling: Just duplicate a random subset
- 2) Synthetic examples using SMOTE (common way)
  - SMOTE (Synthetic Minority Over-sampling Technique)
- Cons: minority data feature space is the same, but dense examples.
  - The model may overfit on the minority class distribution (do bad on anything far from these examples)
- Tip: don't just aim to make them equal size, explore factors of the majority (/ division) or minority (* multiplication)

Oversampling: SMOTE

- SMOTE extends the dataset with synthetic examples as follows:
  - Goal: interpolate between neighbour instances
- Select a random example (A) from our the minority class
- Find k-neighbours of A
- Pick a random neighbour (B) out of the k neighbours
- Pick a random point C on the line segment A==B
  - C ~= A + (B - A) * t [t is random value in range [0-1]
- Extra cons: generated examples might be closer to majority class
- Tip: oversampling must be done only on the train set to avoid data leakage
  - Split to train/val/split, then oversample the train

Oversampling

```
def over_sample(X, y):
    counter = Counter(y)              # Counter({1: 4923, 0: 77})
    factor, majoirty_size = 1, counter[1]  # 4923
    new_sz = int(majoirty_size / factor)

    oversample = SMOTE(random_state=1, sampling_strategy={0: new_sz}, k_neighbors=3)
    #oversample = RandomOverSampler(random_state=1, sampling_strategy={0: new_sz})

    X_os, y_os = oversample.fit_resample(X, y)
    counter = Counter(y_os)                      # Counter({1: 4923, 0: 4923})
    print(counter)

    return X_os, y_os
```

Undersampling majority + Oversampling minority

```
def under_over_sample(X, y):
    counter = Counter(y)              # Counter({0: 77, 1: 4923})
    min_sz = int(counter[1] // 2)
    maj_sz = int(counter[1] // 2)

    oversample = SMOTE(sampling_strategy={0: min_sz}, k_neighbors=5, random_state=1)
    undersample = RandomUnderSampler(sampling_strategy={1: maj_sz}, random_state=1)

    from imblearn.pipeline import Pipeline as imb_Pipeline
    pip = imb_Pipeline(steps=[('over', oversample), ('under', undersample)])

    X_ovs, y_ovs = pip.fit_resample(X, y)
    counter = Counter(y_ovs)
    print(counter)                    # Counter({0: 2461, 1: 2461})

    return X_ovs, y_ovs
```

Cost-Sensitive Learning for Imbalanced Classification

- Cost-sensitive learning is an approach in machine learning where the misclassification costs are different for different classes.
  - E.g. penalty for false negative is different from false positive
- We can use it to assign lower penalty for majority class (or opposite)
  - We can start with penalty: # of minority class examples / # of majority class examples
  - However, we better deal with it as hyperparameter to explore
- **Logistic Regression**: The **loss** function can be **weighted** by the misclassification costs.
  - In SKlearn, pass **classes weights** dictionary (hence we can grid search also)
- Tip: usefulness may vary from an algorithm to another
  - Tree-based algorithms vs logistic regression

Cost-Sensitive Learning for Imbalanced Classification

```
cnter = Counter(y_train)
ir = cnter[0] / cnter[1]    # 0.016

model = LogisticRegression(solver='lbfgs', class_weight={0:1,1:ir})
model.fit(X_train, y_train)

# or for grid search
parameters = {'model__class_weight':
                  [{1: w} for w in [0.01, 0.05, 0.1, 0.5, 1]]}
```

Active Research

- This is an active research problem. From some recent publications
  - Note: i did not read
- Fine-tune based approach
  - Train on something (resampled data) then finetune it (on actual data)
  - Then last model is absorbing again actual distribution
  - I did this trick before to enhance the performance with 2%
- Dynamic sampling
  - During train change percentages of oversampling + undersampling to expose the model more to what it doesn’t see.

Anomaly detection (or outlier detection)

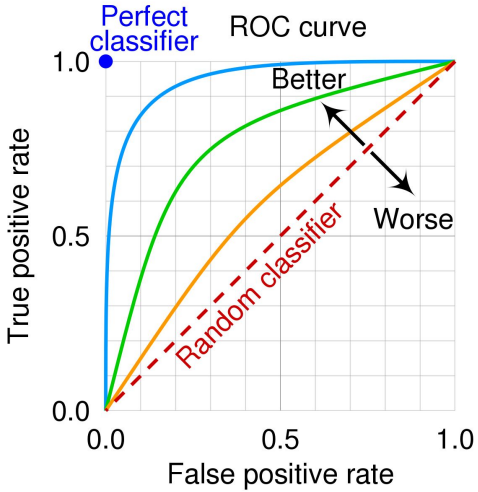
- Anomaly detection (outlier detection) are approaches to identify outliers.
- If we thought of the minority class as an outlier, we may find them.
- Historically, there are many algorithms for that:
  - Statistical Methods (e.g. Percentiles) - Andrew NG
  - Distance-Based Methods (e.g. KNN)
  - Model-Based Methods (e.g. GMM)

Evaluation Metrics

- So far, we learned some approaches to balance the TRAINING part
- What about evaluating the val/test sets?!
- We knew that accuracy and logloss are **misleading** metrics
- We learned about balanced accuracy metric
- What else?

ROC-AUC as metric

- Recall that ROC is mainly based TRP vs FPR
- TPR = Recall = TP / (TP + FN)
- FPR = FP / N
  - N = Negatives = FP + TN
- It is a common advise that this metric is a choice
- However, I don’t think so
- Let’s investigate



ROC-AUC as metric

- Assume P = 1000 and N = 1000000
  - FPR = FP / N
- Add to below, as the measure considers both P and N, high scores could be driven by the majority class misleading us

TP	FP	FPR	Precision	Recall	Notes
75	2000	0.002	0.036	0.075	
75	100	0.0001	0.428	0.075	Huge drop in FP (2000⇒100) is not visible in FPR (~0) Due to large **denominator**: 2000/1000000 vs 100/1000000 On the other side: Precision has visible improvements
150	100	0.0001	0.60	0.150	
300	100	0.0001	0.75	0.300	As P is overall small, small increase in TP leads to big jumps in recall [not severe issue]

Metrics for Imbalanced Datasets

- Given the severe imbalance between P and N, and our attention is mainly getting good P, we need a metric that can observer the positive class
  - Typically imbalanced datasets have limited positive examples
  - Hence, precision based metrics are important here
  - Specifically **F_β-Score and PR-Curve**
- Note that: precision (indicator for FP) and Recall (indicator for FN) covers the 2 possible mistakes. Hence F-Score already has a good overall evaluation
- In practice, we can also in addition to these values for the positive class, we can **switch** the labels and hence report the scores for the majority class reported as positive class
  - As shown in the classification report: we average F-scores of the different classes

Loss for Imbalanced Datasets

- Sometimes, we think how to change the loss itself to handle imbalance datasets
- Focal Loss is tries to handle the class imbalance problem
- It is common in Object Detection task
- One may try it also with other imbalanced data

Cross Validation: StratifiedKFold

- **StratifiedKFold:** Similar to **k-fold cross-validation** but maintains the same class distribution within each fold as it is in the whole dataset.
  - Used with imbalance datasets
  - skf = StratifiedKFold(n_splits=5, random_state=42, shuffle=True)
- **Repeated**StratifiedKFold: It repeats Stratified K-Fold n_repeats times with different randomization in each repetition
  - More robust validation results but computationally expensive

Imbalanced Datasets with Regression problems

- Imbalanced Datasets may appear in regression problems. Examples:
- **Home pricing**: a few houses (rich people) can be very expensive
- **Energy Consumption**: a few houses (rich people) can consumes highly
- **Sales Forecasting**: less sales for niche products
- **Health care bills**: some bills are extremely expensive (severe accidents)
  - Wrong predictions can be highly unwelcomed!
- **Weather Prediction**: in deserts dataset is biased toward non rainy days
- **Customer Lifetime Value Prediction**: a few very high-value customers
- Tackling: you better first identify the data to identify such minority cases
  - Clustinerg? Outlier detection techniques? Perenctiles?

Imbalanced Datasets in Data Preparation

- One interesting source for imbalanced datasets is due to the data preparation process
- Imagine you have multiple sources of data
- Your **selection criteria** discarded many examples/segments ending up making some category rare in the dataset
    - E.g. You only selected examples that has most of the features available

Relevant Materials

- A **Survey** on deep learning with class imbalance
- A **Survey** of Methods for Addressing Class Imbalance in Deep-Learning Based Natural Language Processing

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."