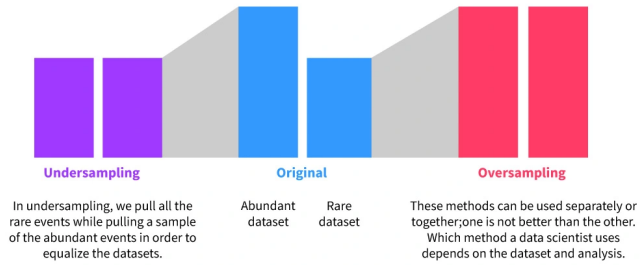
Imbalanced Dataset

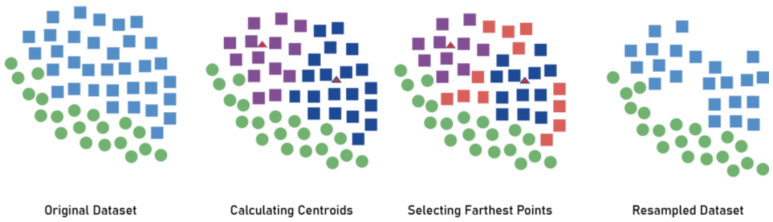
- Definition
  - In classification tasks, number of observation of one of the target class label>>that of other class labels, causing uneven distribution of observations between majority(negative) & minority(positive) class
  - Example in reality: Fraud detection, Quality detection in factory, medical diagnose
- Influence
  - Make the prediction results accuracy of classification model gain bias, which cannot detected by evaluation confusion metrix
- Solutions
  - **1. Enlarge number of samples**
  - **2. Resampling**



    - 1) Undersampling
      - Defintion
        - decrease the majority class sample amount to match up to the numbers of minority class samples
        - use this method when we have huge amout of data
      - Implementation
        - Random Undersampling
          - Defintion: choosing the majority class samples into resampled data is done randomly, limiting the number of majority classes
          - Coding:
            - `from imblearn.under_sampling import RandomUnderSampler`
            - `RUS = RandomUnderSampler(random_state=0,sampling_strategy=1)`
            - `X_rus,y_rus=RUS.fit_resample(X,y)`
        - NearMiss

- Defintion:
  - calculates the distance between all the points in the majority class with the points in the minority class
  - Select n instances of the majority class that have the shortest distance with the minority class.
  - If there are k instances in the minority class, the nearest method will result in k*n instances of the majority class.
- Coding
  - `from imblearn.under_sampling import NearMiss`
  - `NM=NearMiss(sampling_strategy="not minority",n_neighbors=5, version=1)`
    - `version`
      - version1: k closest instances
      - version2: k largest instances
      - version3: for each minority sample, keep m nearest-neighbors. Then, selected majority samples have largest average distance to knn
  - `X_nm,y_nm=NM.fit_resample(X,y)`

- Cluster Centroids



- Defintion: it uses clusters to under-sample majority class data , generating new datasets with K-Means to replace part of majority samples. The minority class stays the same.
- Coding
  - `from imblearn.under_sampling import ClusterCentroids`
  - `CC = ClusterCentroids(sampling_strategy='auto',random_stat`

`e = 1)`

- `X_cc,y_cc=CC.fit_resample(X,y)`

- Tomek Links

- Defintion: pairs of examples of different classes in close vicinity which are near the borderline, then remove the majority elements from the TL
- Coding
  - `!pip install imbalanced-learn`
  - `from imblearn.under_sampling import TomekLinks`
  - `tomek=TomekLinks(sampling_strategy='auto')`
  - `X_tl,y_tl=tomek.fit_resample(X,y)`

- Evaluation
  - Pros: straightforward and easy, reduce storage and running time
  - Cons: lose lots of valuable data. Biased samples are kept by random undersampling method, causing prediction errors.

- 2) Oversampling
  - Defintion
    - Increase the minority class sample amount to match up to the numbers of majority class samples
    - Use this method when we have limited amout of data
  - Implementation
    - SMOTE(Synthetic Minority Oversampling Technique)
      - looks at the feature space for the minority class data and considers its **KNN**.
      - Not simply generate duplicates, but creates synthetic data that are sightly different from the original data point, by slightly moving the data in direction of its neighbor.
      - Coding with imblearn package
        - `from imblearn.over_sampling import SMOTE`
        - `sm=SMOTE(sampling_stategy='auto',random_state=None,k_neignbors=5)`
          - `sampling_stategy`

- float: Ratio $\alpha_{OS} = \frac{N_{rm}}{N_M}$, where N_{rm}: number of samples in minority class after resampling, N_M: the number of samples in majority class
  - Only effective in binary classification problems
- str: specify the class targeted by resampling. At final, the number of samples in the different classes will be equal.
  - `minority` , `not minority` , `not majority` , `all` , `auto` = `not majority`
- dict:
  - keys: the targeted classes
  - value: the desired number of samples for each targeted class.
- callable: functions with y and return dict(keys, values)

- `random_state` : control the randomlization

- `X_sm,y_sm=SMOTE.fit_sample(X,y)`

- Other Variants: SVMSMOTE, SMOTEN, BorderlineSMOTE, KMeansSMOTE

- ADASYN(Adaptive Synthetic Sampling)
  - Defintion
    - Based on SMOTE, additionally add a little variance(more scattered) instead of all the generated samples are linearly correlated with their parents. That means after creating those samples, it addes a random small values to the points.
  - Coding
    - `from imblearn.over_sampling import ADASYN`
    - `adasyn=ADASYN(sampling_stategy='auto',random_state=None)`
    - `X_a,y_a=adasyn.fit_resample(X,y)`
- Random OverSampling
  - Defintion: randomly sample the minority classes and simply duplicate the sampled observations.
  - Coding:
    - `from imblearn.over_sampling import RandomOverSampler`
    - `ROS=RandomOverSampler(random_state=42)`
    - `X_ros,y_ros=ROS.fit_resample(X,y)`
- Evaluation
  - Pros: do not delete valuable information

        - Cons: creates many duplicate data, introducing false information.Higher computation time and more storage are needed. Leads to overfitting.
- **3. Penalize Model When it Misclassifies a Minority**
    - Eg: penalized-SVM, penalized-LDA
    - Defintion: Add weights for minority samples, decrease weights for majority samples. Actually, it generates the new distribution and datasets.
    - Coding with linear regression:
        - `from sklearn.linear_model import LogisticRegression`
        - `LR=LogisticRegression(class_weights={0:1,1:10}`
        - in this case, it add 10 times loss penalize when it misclassifies a minority example. $NewLoss = -10 * ylog(p) - 1 * (1-y)log(1-p)$
- **4. Anomaly Detection**
    - Treat the minority class samples as outliers, change the problem as anomaly detection or change detection
    - use *Isolation forests* or *Autoencoders*
- **5. Ensemble Techniques**
    - Normally, in machine learning ensemble builds multiple estimators on a different randomly selected subset of data.
    - BaggingClassifer: focus on majority data and creates a biased model
    - Use `BalancedBaggingClassifier` in imblearn library
    - Coding
        - `from imblearn.ensemble import BalancedBaggingClassifer`
        - `from sklearn.tree import DecisionTreeClassifer`
        - `BBC=BalancedBaggingClassifer(base_estimator=DecisionTreeClassifier() , sampling_strategy='auto',r eplacement=False, random_state=0)`
        - `BBC.fit(X_train,y_train)`
        - `prediction=BBC.predict(X_test)`
- **6. Use Proper Classification Model**
    - Eg: Change binary classification problem to multi-class classification task
- **7. Change Evaluation Metrics**
    - $Accuracy$ or Error Rates are not work out in imbalanced datasets problem
    - Hence, other evaluation metrices are better in use: Confusion matrix(TP, FN, FP, TN),Precision,Recall, F1, Kappa,ROC and AUC
        - Cohen Kappa
            - Defintion: the level of agreement between the reliability of inter-raters and intra-raters for categorical items.

        - Function: $k = \frac{p_0 - p_e}{1 - p_0}$
            - $p_0 = (TP + TN)/N = Number\ in\ Agreement/N$
            - $p_e = \frac{1}{N^2} \sum_k nk_1 nk_2$
                - Namely, $p_e = [p_e(rater1\ agreement)/N * p_e(rater2\ agreement)/N] + [p_e(rater1\ disagreement)/N * p_e(rater2\ disagreement)/N]$
                    - one rater is classification model, another is real-world observers who know the actual situations
        - range: $[-1, 1]$, normally we care about $[0, 1]$ part, for $k = 0$, no agreement at all, $k = 1$, perfect agreement
        - Calculator: https://idostatistics.com/cohen-kappa-free-calculator/
        - Coding:
            - `from sklearn.metrics import cohen_kappa_score`
            - `cohen_kappa_score(rater1,rater2)`, where rater1 and rater2 are binary pre-defined.
- **8. OHEM(online hard example mining)**
    - select the hard examples which have big effects on classification and detection tasks based on Loss of input ROIs, then train the selected examples in gradient desent. NMS selects the RoIs with largest losses, then removes the ROIs with lower loss RoIs.
    - Applied in **Faster R-CNN**
- **9. Focal Loss(FL)**
    - Also add small amount of easy examples based on method8. At the same time, use balanced cross entroy to balance the classes.
    - Applied in **SSD** model.

- References

    - 炼丹笔记一：样本不均衡问题。https://zhuanlan.zhihu.com/p/56882616
    - 5 Techniques to Handle Imbalanced Data for a Classification Problem. https://www.analyticsvidhya.com/blog/2021/06/5-techniques-to-handle-imbalanced-data-for-a-classification-problem/
    - The five most useful techniques to handle imbalanced datasets. https://medium.com/towards-data-science/the-5-most-useful-techniques-to-handle-imbalanced-datasets-6cdba096d55a
    - imblearn library Document. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html
    - Resampling techniques to handle imbalanced dataset. https://www.kaggle.com/code/prasathm2001/resampling-techniques-to-handle-imbalanced-dataset#Tomek-Links-Under-Sampling

- Review OHEM. https://medium.com/@sh-tsang/review-ohem-training-region-based-object-detectors-with-online-hard-example-mining-object-ad791ad87612

以上内容整理于 幕布文档