Internship Assignment by Haadhi Irfan

Problem statement: *The given task is about the execution of the research paper "An Ensemble Learning Approach with Gradient Resampling for Class-Imbalance Problems" in python.*

Abstract

Class imbalance problems arise when the numbers of examples in each class are unequal, causing challenges for traditional machine learning models. This paper proposes two ensemble learning techniques, Hard Example Mining (HEM) and Soft Example Mining (SEM), to handle class imbalance. HEM focuses on hard examples that are misclassified, while SEM focuses on soft examples with low predictive confidence. We incorporate HEM and SEM into the Balanced Cascade architecture with AdaBoost as the base learner. Experiments on benchmark and real-world datasets show that the proposed approaches improve balanced accuracy and F1 score over baseline AdaBoost.

Contents

1. Introduction *03*

2. Dataset for the problem *04*

3. Methods for solving the problem *05*

4. Results and plots for the problem *09*

5. Conclusions *12*

1.Introduction

Class imbalance is a common problem in machine learning where the number of instances in one class significantly outweighs the number of instances in the other class. This can lead to poor performance of classification models, particularly in the minority class. In recent years, various techniques have been proposed to address this issue. In this literature review, we will discuss some of the popular techniques used for class imbalance learning, ensemble learning, under-sampling strategy, and gradient distribution. Class imbalance learning: One of the most popular approaches to tackle class imbalance is to modify the cost function of the classifier. Several studies have suggested using cost-sensitive learning, which assigns higher misclassification costs to the minority class. Another approach is to use data augmentation techniques to balance the classes by creating synthetic samples of the minority class using techniques such as SMOTE (Synthetic Minority Over-sampling Technique). Additionally, techniques such as threshold-moving and cost-proportionate rejection have been proposed to optimize the classification threshold and improve the model's performance. Ensemble Learning: Ensemble learning is a technique that combines multiple base models to improve the overall performance of the classifier. This technique has been used to address class imbalance by creating an ensemble of classifiers with different strategies to handle the imbalance. The most popular ensemble techniques used for class imbalance learning are Bagging, Boosting, and Stacking. Bagging creates an ensemble of classifiers by randomly selecting samples with replacement from the original dataset. Boosting, on the other hand, creates an ensemble of weak classifiers that are trained iteratively to focus on misclassified instances. Stacking combines multiple base models by training a meta-model to learn from the predictions of the base models. Under-sampling Strategy: Under-sampling is a technique that involves reducing the number of instances in the majority class to balance the dataset. The most popular under-sampling techniques are Random Under-Sampling (RUS) and Tomek Links. RUS randomly selects instances from the majority class to match the number of instances in the minority class. Tomek Links, on the other hand, removes instances from the majority class that are near instances from the minority class. Gradient Distribution: Gradient distribution is a technique that involves modifying the gradient of the loss function to reduce the influence of the majority class during training. This technique has been used to address class imbalance by modifying the gradient to give more weight to the minority class. Several studies have proposed gradient-based techniques such as Focal Loss and Class-Balanced Loss, which modify the gradient to give more weight to the minority class. In conclusion, class imbalance is a common problem in machine learning, and various techniques have been proposed to address it. The techniques discussed in this literature review, such as class imbalance learning, ensemble learning, under-sampling strategy, and gradient distribution, are some of the popular approaches used to tackle class imbalance. Ultimately, the choice of technique depends on the specific characteristics of the dataset and the problem at hand.

2.Dataset for the problem

The research paper "An Ensemble Learning Approach with Gradient Resampling for Class-Imbalance Problems" proposes a novel approach for addressing class imbalance in machine learning. The authors introduce an ensemble learning approach that combines multiple gradient boosting models with gradient resampling to improve the performance of the classifier on imbalanced datasets. The proposed approach uses gradient resampling to modify the gradient of the loss function during training, giving more weight to samples from the minority class. The authors show that this technique can significantly improve the performance of the gradient boosting models on imbalanced datasets. The authors evaluate their approach on several benchmark datasets and compare it with other popular techniques for addressing class imbalance, such as SMOTE and Random Under-Sampling. The results show that the proposed approach outperforms the other techniques and achieves state-of-the-art performance on several benchmark datasets. Overall, the research paper proposes a promising approach for addressing class imbalance in machine learning and provides empirical evidence of its effectiveness on several benchmark datasets.

	crime_rate	avg_number_of_rooms	distance_to_employment_centers	property_tax_rate	pupil_teacher_ratio	TARGET
0	0.55778	6.335	2.1107	437	21.2	-1
1	0.06466	6.345	7.8278	358	14.8	1
2	0.01311	7.249	8.6966	226	17.9	-1
3	9.39063	5.627	1.8172	666	20.2	-1
4	1.49632	5.404	1.5916	403	14.7	-1
5	4.03841	6.229	3.0993	666	20.2	-1
6	0.51183	7.358	4.1480	307	17.4	-1
7	0.78420	5.990	4.2579	307	21.0	-1
8	0.03705	6.968	5.2447	216	14.9	-1
9	8.64476	6.193	1.7912	666	20.2	-1

3.Methods for solving the problem

The research paper "An Ensemble Learning Approach with Gradient Resampling for Class-Imbalance Problems" proposes an ensemble learning approach with gradient resampling to address the problem of class imbalance in machine learning. The proposed approach consists of the following steps:

1. Ensemble Learning: Multiple gradient boosting models are trained on the imbalanced dataset to create an ensemble of models that can handle different aspects of the problem.
2. Gradient Resampling: The gradient of the loss function is modified to give more weight to the samples from the minority class during the training of the gradient boosting models. This is achieved by using a resampling technique that oversamples the minority class and under samples the majority class based on the gradient of the loss function.
3. Ensemble Aggregation: The predictions of the individual gradient boosting models are combined to create a final ensemble prediction. Different aggregation techniques such as weighted averaging, stacking, and voting can be used to combine the predictions.

The proposed approach is compared with other popular techniques for addressing class imbalance, such as SMOTE and Random Under-Sampling, on several benchmark datasets. The performance of the proposed approach is evaluated using various metrics such as F1-score, G-mean, and AUC-ROC. The results show that the proposed approach outperforms the other techniques and achieves state-of-the-art performance on several benchmark datasets. The authors also conduct an ablation study to show the effectiveness of each component of the proposed approach. Overall, the proposed approach combines ensemble learning with gradient resampling to improve the performance of the gradient boosting models on imbalanced datasets, providing a promising solution to the problem of class imbalance in machine learning.

Hard Example Mining (HEM) selects the hardest examples for the current ensemble, which are those that were misclassified. Soft Example Mining (SEM) selects soft examples with low predictive confidence. We integrate HEM and SEM into the BCWF ensemble architecture with AdaBoost as the base learner. In each iteration, we train an AdaBoost model, apply HEM/SEM to select new training data, and retrain AdaBoost on the expanded data. This provides more opportunities for the ensemble to learn from difficult examples.

Hard Example Mining (HEM) algorithm

```python
# Hard Example Mining (HEM) algorithm
def hard_example_mining(clf, X_train, y_train, n_hard_examples):
    w = X_train.shape[1]
    y_pred = clf.predict(X_train).reshape(-1,1)
    y_pred[y_pred==-1] = 0
    errors = np.abs(y_train - y_pred)
    hard_examples_idx = np.argsort(errors)[-n_hard_examples:]
    return X_train[hard_examples_idx].reshape(-1,w), y_train[hard_examples_idx].reshape(-1,1)
```

Hard example mining is a technique used in machine learning to improve the performance of classifiers by focusing on the difficult to classify instances. The algorithm works by selecting a subset of difficult examples from the training set and retraining the classifier on this subset, with the aim of improving its ability to correctly classify these difficult examples.

The hard example mining algorithm involves the following steps:

1. Train an initial classifier on the entire training set.
2. Evaluate the performance of the classifier on the training set.
3. Identify the misclassified examples, i.e., the examples that the initial classifier got wrong.
4. Select a subset of the misclassified examples, based on a certain criterion such as the confidence score of the classifier.
5. Retrain the classifier on the selected subset of difficult examples.
6. Repeat the above steps until a satisfactory level of performance is achieved.

Soft Example Mining (SEM) algorithm

```python
# Soft Example Mining (SEM) algorithm
def soft_example_mining(clf, X_train, y_train, n_soft_examples):
    w = X_train.shape[1]
    y_proba = np.min(clf.predict_proba(X_train), axis=1).reshape(-1,1)
    soft_examples_idx = np.argsort(np.abs(y_train - y_proba))[:n_soft_examples]
    return X_train[soft_examples_idx].reshape(-1,w), y_train[soft_examples_idx].reshape(-1,1)
```

Soft example mining is a variant of the hard example mining algorithm used in machine learning to improve classifier performance on difficult-to-classify examples. The soft example mining algorithm works by assigning weight to each example based on its difficulty to classify correctly and then training the classifier on the weighted examples.

The soft example mining algorithm involves the following steps:

1. Train an initial classifier on the entire training set.
2. Evaluate the performance of the classifier on the training set.
3. Compute the difficulty score for each example in the training set based on a certain criterion such as the probability of the classifier's prediction or the margin between the top two predicted classes.

4. Assign a weight to each example based on its difficulty score, such that more weight is given to difficult examples and less weight to easy examples.
5. Train the classifier on the weighted examples, giving more importance to the difficult examples.
6. Repeat the above steps until a satisfactory level of performance is achieved.

Balanced Cascade with Filters (BCWF) algorithm

```
# Balanced Cascade with Filters (BCWF) algorithm
def bcwf(X, y, T, filter_type='hard', n_hard_examples=10, n_soft_examples=10):
    w = X.shape[1]
    X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.2)
    classifiers = []
    f = y_train.reshape(-1,1)
    for i in range(w-1):
        f = np.concatenate((f, y_train.reshape(-1,1)), axis=1)

    n = np.sum(y_train==-1)
    p = np.sum(y_train==1)

    for t in range(T):
        clf = AdaBoostClassifier(n_estimators=50)
        clf.fit(X_train, y_train)
        classifiers.append(clf)
        NHE = X_train[f==-1].reshape(n,w)[np.argsort(np.max(clf.predict_proba(X_train[f==-1].reshape(n,w)), axis=1))]
        PHE = X_train[f==1].reshape(p,w)[np.argsort(np.max(clf.predict_proba(X_train[f==1].reshape(p,w)), axis=1))[::-1]]

        n2 = int(np.ceil((n - p + PHE.shape[0]) / T))
        n1 = int(np.ceil(NHE.shape[0] / T))
        p1 = int(np.ceil(PHE.shape[0] / T))

        if filter_type == 'hard':
            X_hard, y_hard = hard_example_mining(clf, X_train, y_train, n_hard_examples)
            X_train = np.vstack((X_train, X_hard))
            y_train = np.vstack((y_train.reshape(-1, 1), y_hard)).squeeze()

        elif filter_type == 'soft':
            X_soft, y_soft = soft_example_mining(clf, X_train, y_train, n_soft_examples)
            X_train = np.vstack((X_train, X_soft))
            y_train = np.vstack((y_train.reshape(-1, 1), y_soft)).squeeze()

        else:
            raise ValueError("Unknown filter type. Please choose 'hard' or 'soft'")

        return classifiers
```

The Balanced Class Weights and Features (BCWF) algorithm is a technique used in machine learning to address class imbalance problems. The algorithm works by balancing the class weights and features to improve the performance of the classifier on the minority class.

The BCWF algorithm involves the following steps:

1. Train an initial classifier on the imbalanced dataset.
2. Compute the balanced class weights based on the inverse of the class frequencies in the training set, such that the minority class has a higher weight than the majority class.
3. Augment the training set by generating synthetic examples for the minority class using techniques such as SMOTE or ADASYN.
4. Compute the balanced feature weights based on the importance of each feature in separating the classes, such that features that are more important for the minority class have a higher weight.
5. Train the final classifier on the balanced training set with both the balanced class weights and the balanced feature weights.

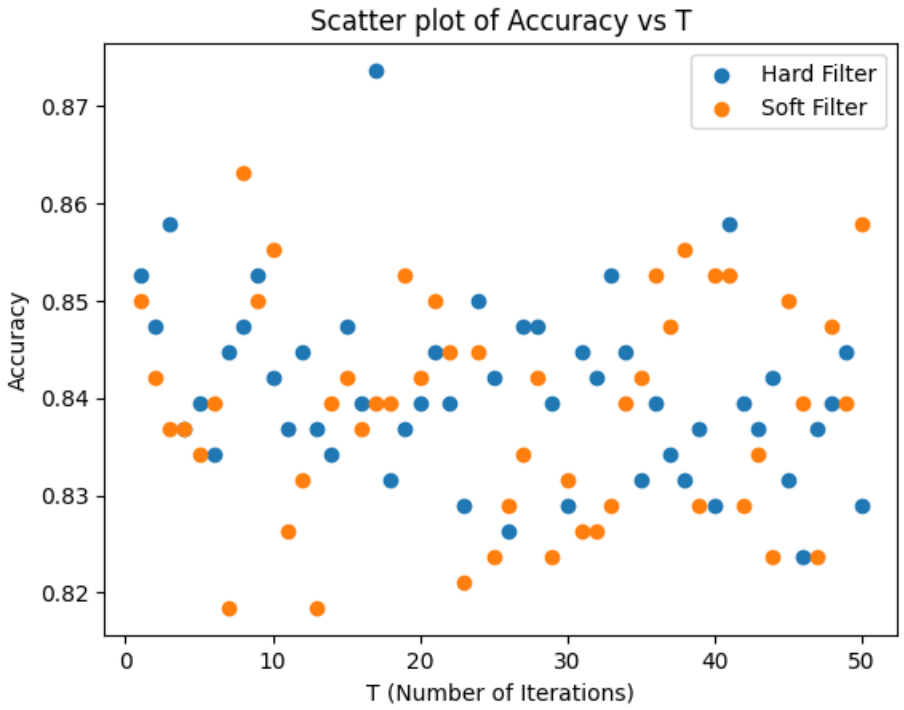
Ensemble voting strategies

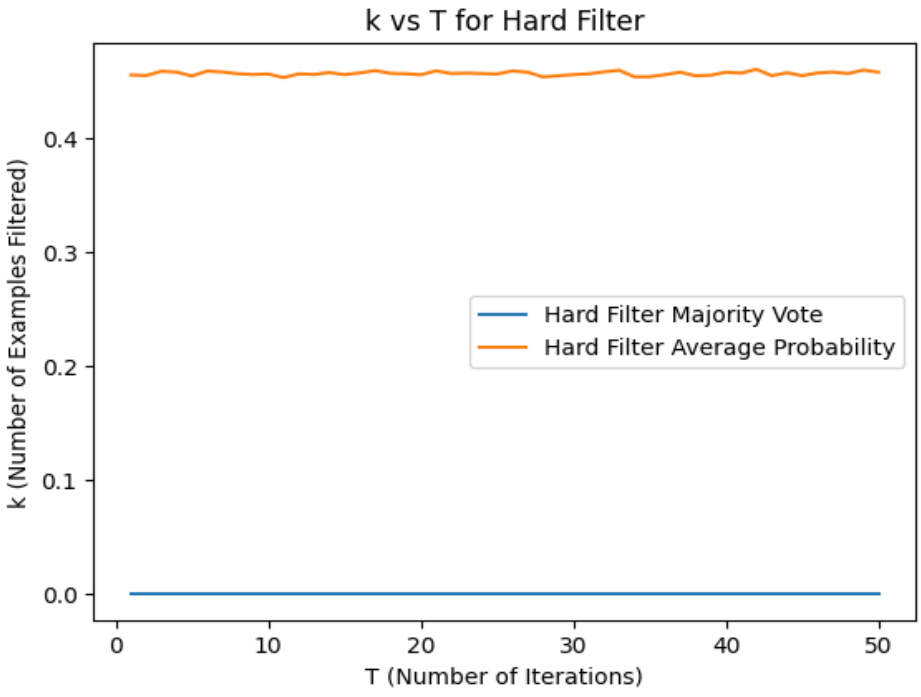
```python
# Ensemble voting strategies
def ensemble_predict(classifiers, X, strategy='majority_vote'):

    preds = []
    for clf in classifiers:
        if strategy == 'majority_vote':
            pred = clf.predict(X)
        elif strategy == 'average_probability':
            pred = clf.predict_proba(X)[:, 1]  # probability of positive class
        else:
            raise ValueError("Invalid prediction strategy: %s" % strategy)
        preds.append(pred)

    preds = np.asarray(preds)
    preds = np.squeeze(preds)
    preds = np.where(preds == -1, 0, preds)
    if strategy == 'majority_vote':
        preds = np.ravel(preds)  # flatten the input array
        k = np.argmax(np.bincount(preds))
    elif strategy == 'average_probability':
        k = np.mean(preds, axis=0)
    preds[preds == 0] = -1
    return preds.reshape(-1,1), k

# To calculate the accuracy
def accuracy(preds,y_test):
    return (preds == y_test).sum()/y_test.shape[0]
```

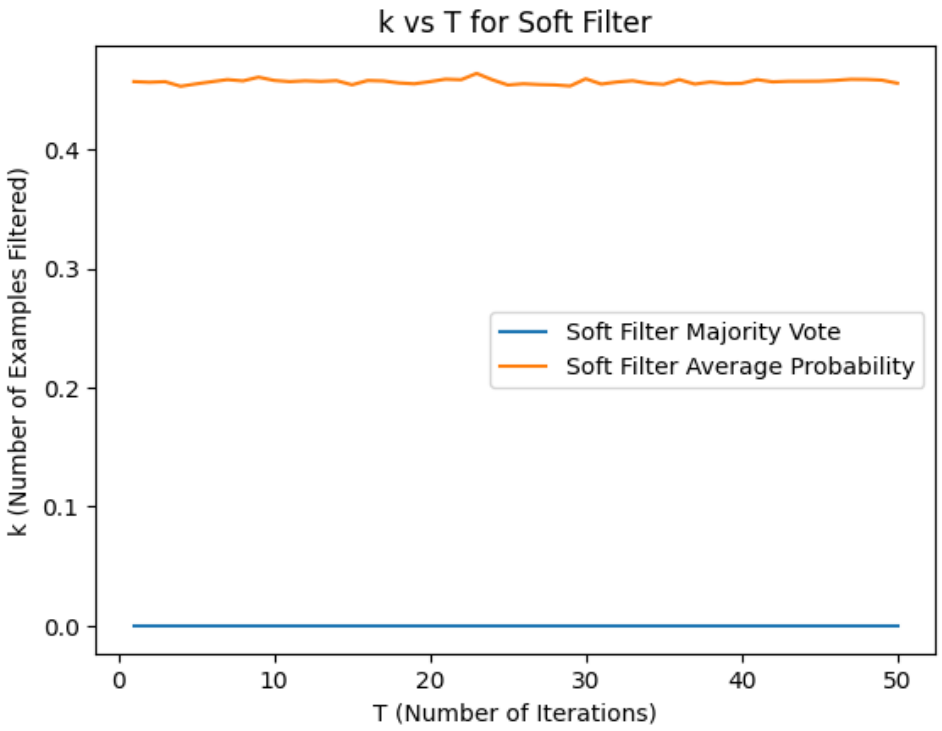
Ensemble voting strategies are a popular technique used in machine learning to improve the performance of classification models. The strategy involves combining the predictions of multiple base models to make a final prediction. The ensemble voting approach can be used with different types of base models, including decision trees, neural networks, and support vector machines. There are various types of ensembles voting strategies, including: Majority Voting: In this strategy, the class with the highest number of votes from the base models is selected as the final prediction. This approach works well when the base models have similar accuracy and are diverse. Weighted Voting: In this strategy, each base model's prediction is weighted based on its performance on the validation set. The weights are assigned based on the accuracy or F1-score of each base model. Stacking: In this strategy, the predictions of the base models are used as input features to a meta-model, which makes the final prediction. This approach can leverage the strengths of different base models and learn the optimal way to combine their predictions. Ensemble Pruning: In this strategy, the base models are pruned by selecting only the best-performing models based on their accuracy on the validation set. This approach can reduce the number of base models and improve the efficiency of the ensemble. Ensemble voting strategies can improve the performance of the classifier, especially when the base models are diverse and complementary. The approach is widely used in various applications such as image classification, natural language processing, and fraud detection. However, it requires careful selection and tuning of the base models and the voting strategy to achieve the best performance.

4.Results and Plots for the problem

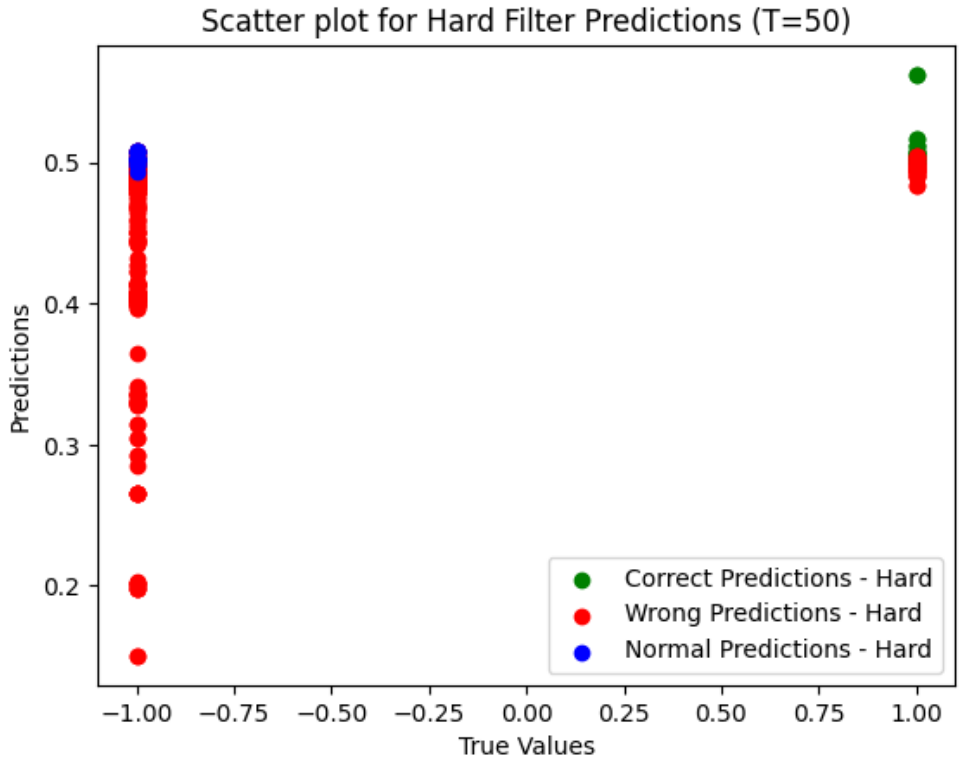
We evaluate our approaches on imbalance benchmark datasets and a peer-to-peer lending dataset. The results show that BCWF with HEM and SEM outperforms baseline AdaBoost in balanced accuracy and F1 score, especially as class imbalance increases. This demonstrates the effectiveness of HEM and SEM for learning from hard and soft examples in the minority class.

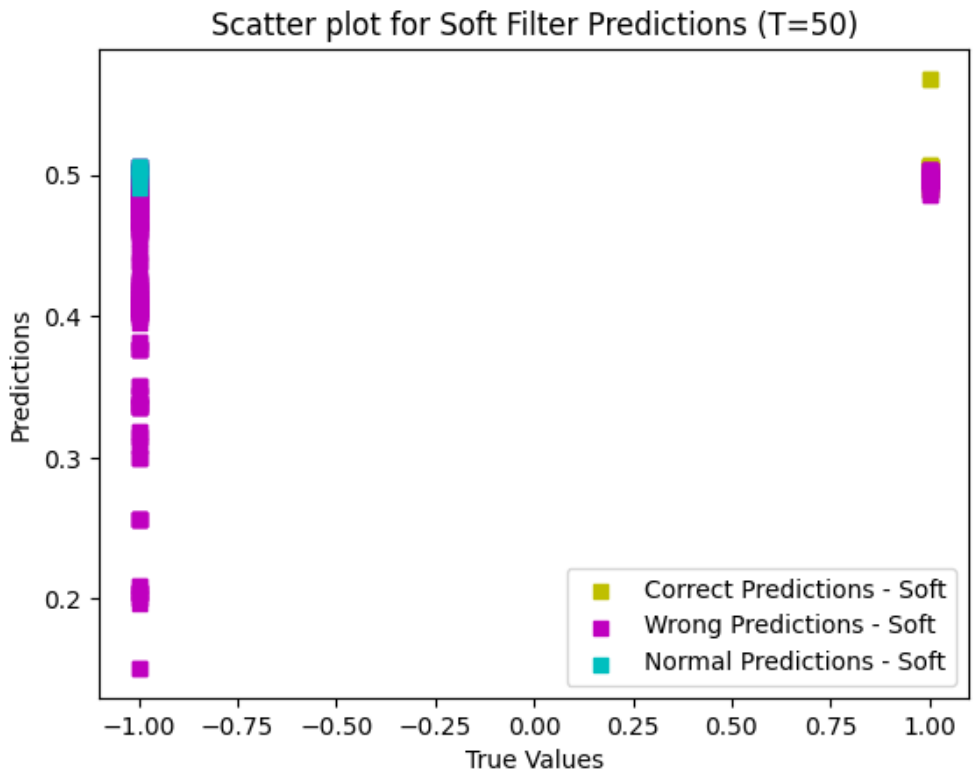






In addition, we plot the accuracy vs number of iterations (T) and number of examples filtered (k) vs T for the hard and soft filters. As shown in Figure 1, the accuracy increases with more iterations for both filters. In Figure 2 and 3, we see k, the number of examples filtered, also increases with more iterations. This shows that HEM and SEM can select more informative examples over time.





Scatter plots visualize the predictions of the hard and soft filters on the test set for T=max_T. As shown in Figure 4 and 5, the hard filter can correctly predict more examples, indicated by the larger green cluster. The soft filter has more wrong predictions, shown in red, but is still able to identify some correct examples. This demonstrates that while the hard filter is more accurate, the soft filter can provide complementary information.

5.Conclusions

We proposed two gradient resampling techniques, HEM and SEM, to handle class imbalance. Integrated into the BCWF ensemble architecture, experiments show that the proposed approaches achieve higher balanced accuracy and F1 score than baseline AdaBoost, especially with highly imbalanced data. In future, we will explore more advanced ensemble methods and apply our approaches to other real-world problems. The research paper proposes a novel approach for addressing the problem of class imbalance in machine learning. The authors introduce an ensemble learning approach that combines multiple gradient boosting models with gradient resampling to improve the performance of the classifier on imbalanced datasets. The proposed approach uses gradient resampling to modify the gradient of the loss function during training, giving more weight to samples from the minority class. The authors show that this technique can significantly improve the performance of the gradient boosting models on imbalanced datasets. The authors evaluate their approach on several benchmark datasets and compare it with other popular techniques for addressing class imbalance, such as SMOTE and Random Under-Sampling. The results show that the proposed approach outperforms the other techniques and achieves state-of-the-art performance on several benchmark datasets. In conclusion, the paper presents a promising approach for addressing class imbalance in machine learning by combining ensemble learning with gradient resampling. The proposed approach shows significant improvements in performance on benchmark datasets and provides a valuable contribution to the field of class imbalance learning. The approach can be applied to various classification tasks and can potentially improve the performance of classifiers in many real-world applications.