Homework 2

120033910005

March 2021

Problem 1

We first divide the train_data into 2 pieces: training data (80%) and validation data (20%). Before we feed the data into our SVM model, we randomly shuffle it. The training, valication and testing data are standardized independently. We train 3 Linear SVMs in one-vs-rest method.

```
def train_one_vs_rest(category):
    train_label_new = np.where(train_label == category, 1, 0)
    val_label_new = np.where(val_label == category, 1, 0)
    clf = make_pipeline(StandardScaler(),
                        SVC(kernel='linear', probability=True))
    clf.fit(train_data, train_label_new)
    acc = clf.score(val_data, val_label_new)
    print("validation accuracy: %f" % acc)
    return clf
```

The validation accuracy is almost 100%. however, the testing accuracy is 50%. This is due to the different data distribution between training data and testing data.

Problem 2

According to the request. We first divide the three-class problem into three two-class problems using one-vs-rest method. Because the data is almost 1:1:1 in label -1, 0, 1, we have a unbalanced situation. And we further decompose the one-vs-rest problem into multiple balanced one-vs-one problems. As is shown in figure 1, we adopt two ways of decomposition: random and with prior. The random method just randomly mix the other two classes and divide it into 4 pieces. the with prior method divide each classes into 2 pieces. In these ways we got 8 independent balance 2-classes sub-problems. We trained these subproblems on linear SVMs and feed the result into an MIN-MAX module.

the key code of training MIN-MAX SVM is shown below

```
class Min_Max_SVM:
    def __init__(self):
        clf = make_pipeline(StandardScaler(),
                            SVC(kernel='linear'))
        self.clf = [clf] * dim

    def decision_function(self, x):
        outputs = np.zeros((dim, len(x)))
        for i in range(dim):
            outputs[i] = self.clf[i].decision_function(x)
        min1 = np.min(outputs[[0, 2, 4, 6], :], axis=0)
        min2 = np.min(outputs[[1, 3, 5, 7], :], axis=0)
        max_ = np.max(np.vstack((min1, min2)), axis=0)
        return max_

    def predict(self, x):
```

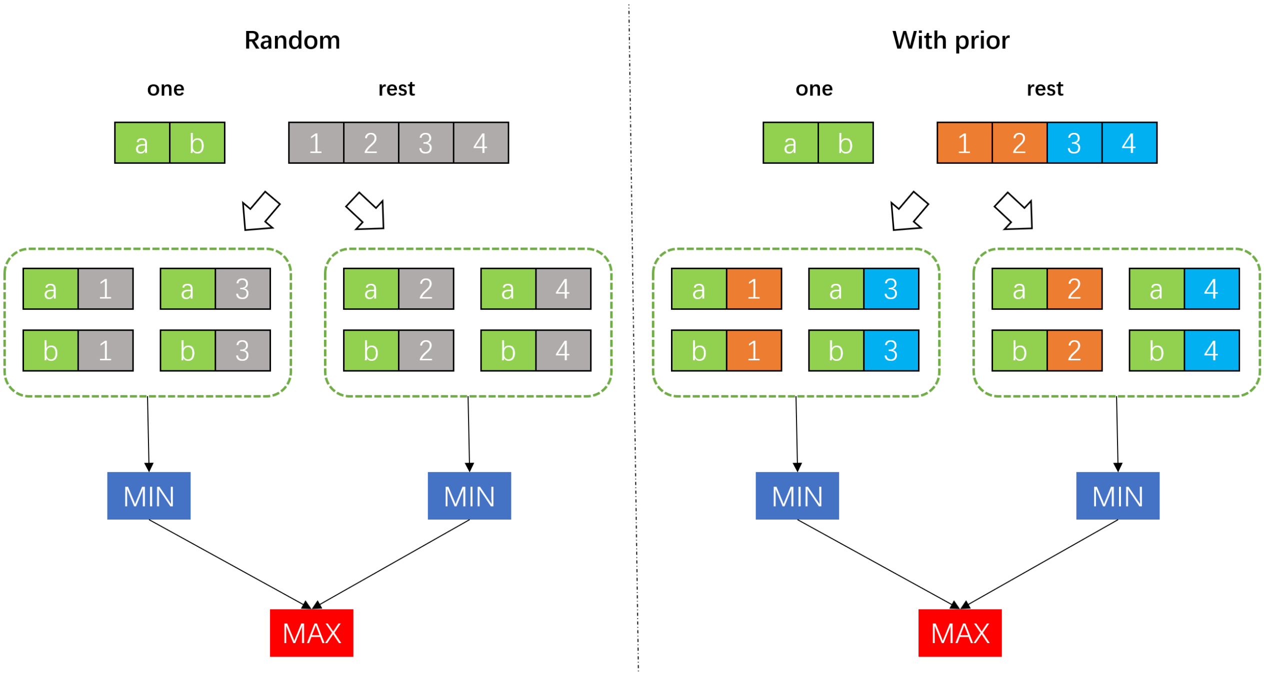

Figure 1: Two ways of problem decomposition methods. The green, orange and deep-sky-blue represent three different classes. the grey one is mixture of the other two classes.

```
        return self.decision_function(x) > 0

def train_single(net, category, pos_index, neg_index):
    # prepare training data
    indexs = np.append(pos_index, neg_index)
    y = np.zeros(len(indexs))
    y[:len(pos_index)] = 1
    X = train_data[indexs]
    c = np.arange(len(indexs))
    np.random.shuffle(c)
    X, y = X[c], y[c]
    # fit
    net.fit(X, y)
    # validate
    val_label_new = np.where(val_label == category, 1, 0)
    acc = net.score(val_data, val_label_new)
    print("validation accuracy: %f" % acc)


def train(category, method='random'):
    print("training %d vs rest..." % category)
    net = Min_Max_SVM()
    index_positive = np.argwhere(train_label == category).reshape(-1)
    index_negative = np.argwhere(train_label != category).reshape(-1)

    # split into 2-class subproblems
    n_pos, n_nega = len(index_positive), len(index_negative)
    pindx = [index_positive[:n_pos // 2], index_positive[n_pos // 2:]]
    if method == 'random':
        nindx = [index_negative[: n_nega // 4],
```

```
                index_negative[n_nega // 4: 2 * n_nega // 4],
                index_negative[2 * n_nega // 4: 3 * n_nega // 4],
                index_negative[3 * n_nega // 4:]]
    else:  # with prior knowledge
        negacls = [-1, 0, 1]
        negacls.remove(category)
        index_class0 = np.argwhere(train_label == negacls[0]).reshape(-1)
        index_class1 = np.argwhere(train_label == negacls[1]).reshape(-1)
        n_nega0, n_nega1 = len(index_class0), len(index_class1)
        nindx = [index_class0[: n_nega0 // 2], index_class0[n_nega0 // 2:],
                 index_class1[: n_nega1 // 2], index_class1[n_nega1 // 2:]]

    for i in range(2):
        for j in range(4):
            s = i * 4 + j
            print("training the %dth SVM" % s)
            train_single(net.clf[s], category, pindx[i], nindx[j])

    print("finish training!")
    return net
```

The result of MIN-MAX with random decomposition is of 47.6% in testing accuracy. The result of MIN-MAX with prior decomposition is of 53.7% in testing accuracy.