Homework 7 for EE559
Author: Chengyao Wang
USCID: 6961599816
Contact: chengyao@usc.edu
Topic: Support Vector Machines

Results and Brief Discussion:

(a)[Rbf+OvR+L2]:

Results:

Best Model with RBF L2 Penalized [Hamming Loss, Penalty, Kernel Attributes]:
Label 1: [0.007742083697420191, 7.38905609893065, 2.0]
Label 2: [0.009928328046201423, 7.38905609893065, 1.6]
Label 3: [0.009306536713306498, 20.085536923187668, 1.9000000000000001]
Test Result:
Label 1 (Test Error):
Prediction: [18.0, 160.0, 648.0, 1333.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [20.0, 158.0, 640.0, 1341.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.010189902732746642
Label 1 (Train Error):
Prediction: [48.0, 384.0, 1525.0, 3079.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [48.0, 384.0, 1525.0, 3079.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.0003971405877680699
Label 2 (Test Error):
Prediction: [1261.0, 160.0, 91.0, 475.0, 74.0, 34.0, 18.0, 46.0, 0.0, 0.0, 0.0]
True: [1269.0, 158.0, 89.0, 474.0, 72.0, 32.0, 20.0, 45.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.012042612320518759
Label 2 (Train Error):
Prediction: [2881.0, 384.0, 221.0, 1120.0, 198.0, 81.0, 48.0, 103.0, 0.0, 0.0, 0.0]
True: [2881.0, 384.0, 221.0, 1119.0, 198.0, 82.0, 48.0, 103.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.0005957108816521049
Label 3 (Test Error):
Prediction: [201.0, 1059.0, 161.0, 92.0, 151.0, 324.0, 74.0, 34.0, 17.0, 46.0, 0.0]
True: [204.0, 1065.0, 158.0, 89.0, 146.0, 328.0, 72.0, 32.0, 20.0, 45.0, 0.0]
Hamming Loss: 0.012968967114404817
Label 3 (Train Error):
Prediction: [468.0, 2413.0, 384.0, 221.0, 326.0, 793.0, 198.0, 82.0, 48.0, 103.0, 0.0]
True: [468.0, 2413.0, 384.0, 221.0, 326.0, 793.0, 198.0, 82.0, 48.0, 103.0, 0.0]
Hamming Loss: 0.0

Discussion:

This multi-label & multi-class classification problem is divided into three independent one-multi-class label problem here, and is solved by SVM. Since it's a Ridge-Penalized RBF Kernel SVM, cross validation must be performed on two hyperparameters: ridge penalty C, free kernel parameter σ. The range of the two parameters on which CVed are:

penaltyList = np.exp(range(-3, 6))
sigmaList = np.arange(0.1, 2.1, 0.1)

There is a combination of 200 parameters, and hamming loss/score of each are compared for choosing the best model. The Hamming loss is the fraction of labels that are incorrectly predicted, while in this scenario is equavilant to prediction error since the problem is separated into 3 single label problems.

Cross validation packages in Sklearn cannot satisfy this scenario, so CV process is re-designed. ***Sklearn.preprocessing.StratifiedKFold*** is used to split the primal training data set into 10 folds, each with the shape of $\#_{\text{instances}} \times 22$. In each iteration of the cross validation, 1 of the 10 folds take turns to be the validation set, and the rest are concatenate into the training set before fitting into the estimator defined by the hyperparameters in the current iteration. Take average on the hamming loss and choose the best model when it's at its minimum.

Also, ***coreEstimator*** has to be wrapped into ***sklearn.multiclass.OneVsRestClassifier*** to conduct the classification process in the OvR manner, because kernel SVM is OvO in default.

coreEstimator = SVC(C=C), kernel='rbf', gamma=sigma)
OvRestimator = OneVsRestClassifier(coreEstimator)
OvRestimator.fit(self.trainData, self.yTrain[:, 0])

The result shows that kernel SVM performs well in this situation. The train error is slightly smaller than test error, which is expected in most machine learning cases. The best average score during CV is also shown above, and its slightly lower than test score because of the correlatedness among the ten 'different' traing datasets during CV, but still it's a rather good test score prediction.

Theortically, optimal width of the RBF kernel is a good indicator of the variance of the data set, and they're postively related, i.e. the smaller the optimal RBF kernel width, the smaller the dataset's variance. Actually, the penalty also has a similar property. However, the raw dataset implemented here is already centeralized and normalized by the provider, so there's no point to discuss here.

(b)[Linear+OvR+L1]:

Results:

Best Model with Linear L1 Penalized [Hamming Loss, Penalty]:
Label 1: [0.06512631728995809, 7.38905609893065]
Label 2: [0.05181075296983158, 54.598150033144236]
Label 3: [0.04389161273429434, 7.38905609893065]
Test Result:
Label 1 (Test Error):
Prediction: [0.0, 161.0, 651.0, 1347.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [20.0, 158.0, 640.0, 1341.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.06947660954145438
Label 1 (Train Error):
Prediction: [1.0, 393.0, 1507.0, 3135.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [48.0, 384.0, 1525.0, 3079.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.06374106433677522

Label 2 (Test Error):
Prediction: [1282.0, 162.0, 67.0, 511.0, 70.0, 14.0, 9.0, 44.0, 0.0, 0.0, 0.0]
True: [1269.0, 158.0, 89.0, 474.0, 72.0, 32.0, 20.0, 45.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.05419175544233441
Label 2 (Train Error):
Prediction: [2954.0, 393.0, 165.0, 1176.0, 181.0, 47.0, 18.0, 102.0, 0.0, 0.0, 0.0]
True: [2881.0, 384.0, 221.0, 1119.0, 198.0, 82.0, 48.0, 103.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.04924543288324067
Label 3 (Test Error):
Prediction: [207.0, 1066.0, 159.0, 76.0, 148.0, 355.0, 72.0, 16.0, 14.0, 46.0, 0.0]
True: [204.0, 1065.0, 158.0, 89.0, 146.0, 328.0, 72.0, 32.0, 20.0, 45.0, 0.0]
Hamming Loss: 0.05048633626679018
Label 3 (Train Error):
Prediction: [481.0, 2436.0, 390.0, 173.0, 337.0, 830.0, 189.0, 48.0, 43.0, 109.0, 0.0]
True: [468.0, 2413.0, 384.0, 221.0, 326.0, 793.0, 198.0, 82.0, 48.0, 103.0, 0.0]
Hamming Loss: 0.03971405877680699

Discussion:

Just a switch of the **coreEstimator** with the following compared with (b):

coreEstimator = LinearSVC(penalty = 'l1', C = C, dual=False, max_iter=100000)
OvRestimator = OneVsRestClassifier(coreEstimator)
OvRestimator.fit(CVtraindata, CVtrainlabel)
hammingScore = hamming_loss(self.yTest[:, 0], OvRestimator.predict(self.testData))

L1-penalized, aka Lasso, implicitly mutes some of the features, thus serves as a natural way of feature selection. The conflict between Lasso and the guiding ideology of kernel functions in SVM makes the combination of these two hardly used. Thus, linear kernel is used in this scenario and it can be noticed that the hamming loss is much higher than that of the RBF kernel but still can be considered not so bad. The simplest reason for this is owe to the non-linear-separability of the most valuable subset of the features, since Lasso is used. RBF kernel possesses an infinite VC dimension and can be safely concluded to be better by its better performance on the test dataset.

Also, we can notice that, the number of valuable features for predicting label 2 is smaller than that used in predicting the other two, because optimal penalty for label 2 given by CV is the highest, which will mute more features.

(c)[Linear+OvR+L1+SMOTE]

Result:

Label 1 (Hamming Loss, Penalty):
Before Smote: Counter({3.0: 3079, 2.0: 1525, 1.0: 384, 0.0: 48})
After Smote: Counter({3.0: 3079, 1.0: 3079, 2.0: 3079, 0.0: 3079})
[0.05626559922162527, 2.718281828459045]
Label 2 (Hamming Loss, Penalty):

Before Smote: Counter({0.0: 2881, 3.0: 1119, 1.0: 384, 2.0: 221, 4.0: 198, 7.0: 103, 5.0: 82, 6.0: 48})
After Smote: Counter({0.0: 2881, 1.0: 2881, 2.0: 2881, 3.0: 2881, 4.0: 2881, 5.0: 2881, 6.0: 2881, 7.0: 2881})
[0.04733816320645905, 148.4131591025766]
Label 3 (Hamming Loss, Penalty):
Before Smote: Counter({1.0: 2413, 5.0: 793, 0.0: 468, 2.0: 384, 4.0: 326, 3.0: 221, 6.0: 198, 9.0: 103, 7.0: 82, 8.0: 48})
After Smote: Counter({0.0: 2413, 2.0: 2413, 1.0: 2413, 3.0: 2413, 4.0: 2413, 5.0: 2413, 6.0: 2413, 7.0: 2413, 8.0: 2413, 9.0: 2413})
[0.041852988580638524, 20.085536923187668]
Test Result:
Label 1 (Test Error):
Prediction: [0.0, 160.0, 653.0, 1346.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [20.0, 158.0, 640.0, 1341.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.06947660954145438
Label 1 (Train Error):
Prediction: [1.0, 393.0, 1507.0, 3135.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [48.0, 384.0, 1525.0, 3079.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.06294678316123908
Label 2 (Test Error):
Prediction: [1280.0, 162.0, 77.0, 481.0, 81.0, 19.0, 12.0, 47.0, 0.0, 0.0, 0.0]
True: [1259.0, 160.0, 92.0, 450.0, 95.0, 34.0, 24.0, 45.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.05419175544233441
Label 2 (Train Error):
Prediction: [2954.0, 391.0, 159.0, 1208.0, 163.0, 43.0, 18.0, 100.0, 0.0, 0.0, 0.0]
True: [2891.0, 382.0, 218.0, 1143.0, 175.0, 80.0, 44.0, 103.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.04805401111993646
Label 3 (Test Error):
Prediction: [207.0, 1066.0, 160.0, 74.0, 149.0, 354.0, 72.0, 16.0, 15.0, 46.0, 0.0]
True: [204.0, 1065.0, 158.0, 89.0, 146.0, 328.0, 72.0, 32.0, 20.0, 45.0, 0.0]
Hamming Loss: 0.04909680407596109
Label 3 (Train Error):
Prediction: [480.0, 2436.0, 390.0, 174.0, 336.0, 829.0, 189.0, 48.0, 46.0, 108.0, 0.0]
True: [468.0, 2413.0, 384.0, 221.0, 326.0, 793.0, 198.0, 82.0, 48.0, 103.0, 0.0]
Hamming Loss: 0.03951548848292295

Discussion:

A SMOTE preprocessing step is added before splitting the data, and is realized by:

```
from imblearn.over_sampling import SMOTE
smModel = SMOTE()
resampledData, resampledLabel = smModel.fit_resample(self.trainData, ylabel)
print('Before Smote:', collections.Counter( np.ndarray.tolist(ylabel) ) )
print('After Smote:', collections.Counter( np.ndarray.tolist(resampledLabel) ) )
```

And the rest is indentical to (c). From the test scores of the three labels, we can see that SMOTE has a limited influence in the prediction of label 1&2. However, for label 3, SMOTE

improved it by 0.10. Part of the reason might be the up/over-sampling strategy in SMOTE can only handle the bias introduced by data imbalance, rather than other more important aspects like the orientation of the decision hyperplanes. If the data points are sparse around the boundary, calibrating the bias will do little help. Also, though linear kernel performs worse than RBF kernel, a score of 0.05 is never a bad result numerically. According to the definition of hinge loss in SVM optimization target function, many, nearly all of them produces 0 loss and linear combinations of them will also produce 0 loss. Thus, SMOTE may suffer from low efficiency when implemented in easily classified datasets, especially together with linear kernels since up-sampling is also linear[1]. Gaussian kernel is redone, and the results are the following:

(d)[Classifer Chain]

Result:

Best Chain Classifier Model Under HammingScore (Hamming Score, Penalty):
Label 1: [0.06512631728995809, 7.38905609893065]
Label 2: [0.0113109507997832, 148.4131591025766]
Label 3: [0.003571162055746495, 148.4131591025766]
Test Result:
Label 1 (Train Error):
Prediction: [1.0, 166.0, 627.0, 1365.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [24.0, 160.0, 621.0, 1354.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.06808707735062529
Label 1 (Test Error):
Prediction: [0.0, 393.0, 1522.0, 3121.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
True: [44.0, 382.0, 1544.0, 3066.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.06493248610007943
Label 2 (Train Error):
Prediction: [2891.0, 382.0, 220.0, 1155.0, 175.0, 65.0, 45.0, 103.0, 0.0, 0.0, 0.0]
True: [2891.0, 382.0, 218.0, 1143.0, 175.0, 80.0, 44.0, 103.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.004765687053216839
Label 2 (Test Error):
Prediction: [1283.0, 165.0, 63.0, 468.0, 83.0, 46.0, 2.0, 49.0, 0.0, 0.0, 0.0]
True: [1259.0, 160.0, 92.0, 450.0, 95.0, 34.0, 24.0, 45.0, 0.0, 0.0, 0.0]
Hamming Loss: 0.07642427049559981
Label 3 (Train Error):
Prediction: [471.0, 2420.0, 382.0, 218.0, 328.0, 815.0, 175.0, 80.0, 44.0, 103.0, 0.0]
True: [471.0, 2420.0, 382.0, 218.0, 328.0, 815.0, 175.0, 80.0, 44.0, 103.0, 0.0]
Hamming Loss: 0.0
Label 3 (Test Error):
Prediction: [209.0, 1073.0, 165.0, 63.0, 152.0, 321.0, 81.0, 46.0, 2.0, 47.0, 0.0]
True: [201.0, 1058.0, 160.0, 92.0, 144.0, 306.0, 95.0, 34.0, 24.0, 45.0, 0.0]
Hamming Loss: 0.07549791570171376

[1] https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html

Discussion:

The labels are in 1 -> 2 -> 3 order. A better way of performing chain classifiers actually, pretty much like recursive feature selection method, is recursively adding the best predicted label under the current model to the feature space and end upon exhaustion of features. The best can be defined in arbitrary metrics, including error rate, f1, precision etc. Here, its naïve version is used - by predetermining the order. The general process of Classification Chain is:

1. Predict the label that comes first under the original feature space.
2. One-hot-code the true value of that label in train dataset, and the previous prediction.
3. Cancatenate the former into training dataset, the latter into test dataset.
4. Train the next classifier with the augmented training data, and make predictions.
5. Iterate until all features are classified.

The result presented here is perfectly against common sense that using more features will always lead to better results, especially Lasso is here to mute the added features if they are unvaluable. Do some cheats on the coded features by substituting the previous prediction with coded true test label in step (3), and can get:

Used Correct Label (Cheated):

Train Hamming Loss:
1st Classifier: 0.06808707735062529
2nd Classifier: 0.004765687053216839
3rd Classifier: 0.0
Test Hamming Loss:
1st Classifier: 0.06493248610007943
2nd Classifier: 0.010189902732746642
3rd Classifier: 0.003242241778601204

The classifier chain improved its performance drastically, is now much close to the scores returned by CV, meanwhile also proving that the labels are somehow correlated. The learning process of CC violates a key assumption of supervised learniang that train data and test data are indentically distributed[2]. Though in CC, the train dataset and test dataset both introduced another (or few others if in multi-class with one-hot-coded) label to help in classifying the target label, the extended feature plugged into the test dataset is actually an estimate of the label plugged in, while train dataset accepted the true value. And the result of such is explicitly shown above, increasing the loss from 0.5x to 0.7x. This problem can be seen as a problem of *attribute noise*, which intuitively speaking - *clean training data vs. noisy test data*.[3] So we have to pay more attention when applying Classifier Chains.

[2] Rectifying Classifier Chains for Multi-Label Classification∗

[3] Class Noise vs. Attribute Noise: A Quantitative Study of Their Impacts