Out[1]: Toggle Code

Assignment 4

Chen Xupeng

Ex 1.

(10 pt) Use the Strong Duality Theorem to show that

$$\min_{\beta_0,\beta} \sum_{i=1}^{n} \left(y - \beta_0 - x^T\beta\right)^2$$
$$\text{s.t. } \sum_{j=1}^{p} |\beta_j| \leq s$$

where s > 0, is equivalent to the unconstrained problem

$$\min_{\beta_0,\beta} \sum_{i=1}^{n} \left(y - \beta_0 - x^T\beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$
$$\text{for some } \lambda > 0.$$

primal problem:

$$\min_{\beta \cdot \beta_0} \sum_{i=1}^{n} \left(y - \beta_0 - x^T\beta\right)^2$$
$$s.t. \sum_{j=1}^{p} |\beta_j| \leq s$$

dual problem:

$$\max_{\lambda} \theta(\lambda) \inf_{\beta \cdot \beta_0} \{ \sum_{i=1}^{n} \left(y - \beta_0 - x^T\beta\right)^2 + \lambda (\sum_{j=1}^{p} |\beta_j| - s) \}$$
$$\text{st. } \lambda \geqslant 0$$

assume the optimal satisfies when $\lambda = \lambda_0$:

$$\inf_{\beta \cdot \beta_0} \{ \sum_{i=1}^{n} \left(y - \beta_0 - x^T\beta\right)^2 + \lambda_0 (\sum_{j=1}^{p} |\beta_j| - s) \}$$

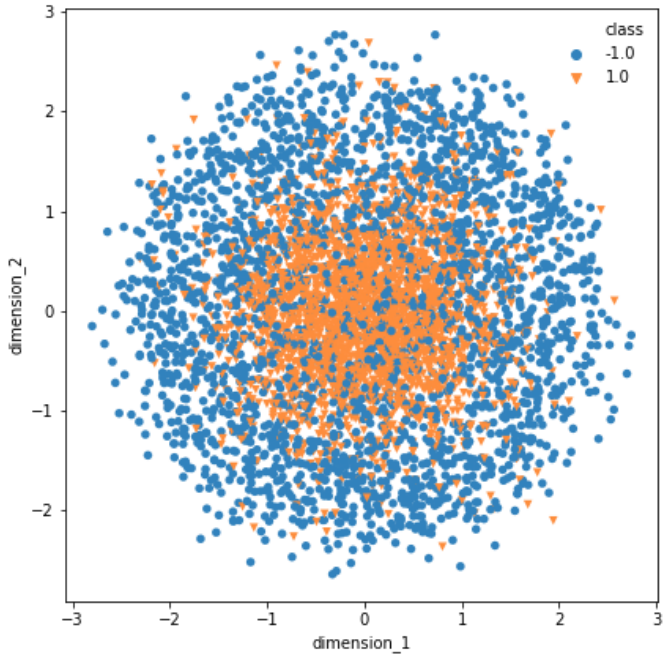
according to Strong Duality Theorem it is also the optimal for primal problem, omitting the constant, we can have minimizing the original is equal to minimizing:

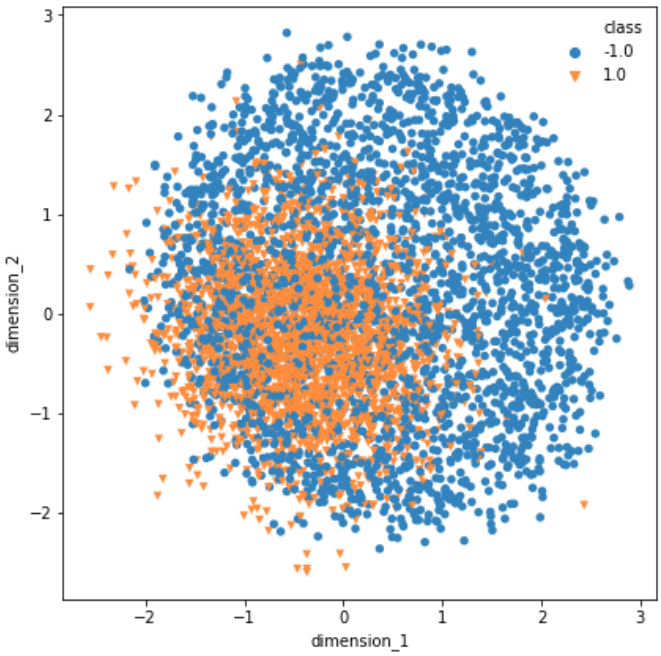
$$\min_{\beta_0,\beta} \sum_{i=1}^{n} \left(y - \beta_0 - x^T\beta\right)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$
$$\text{for some } \lambda > 0.$$

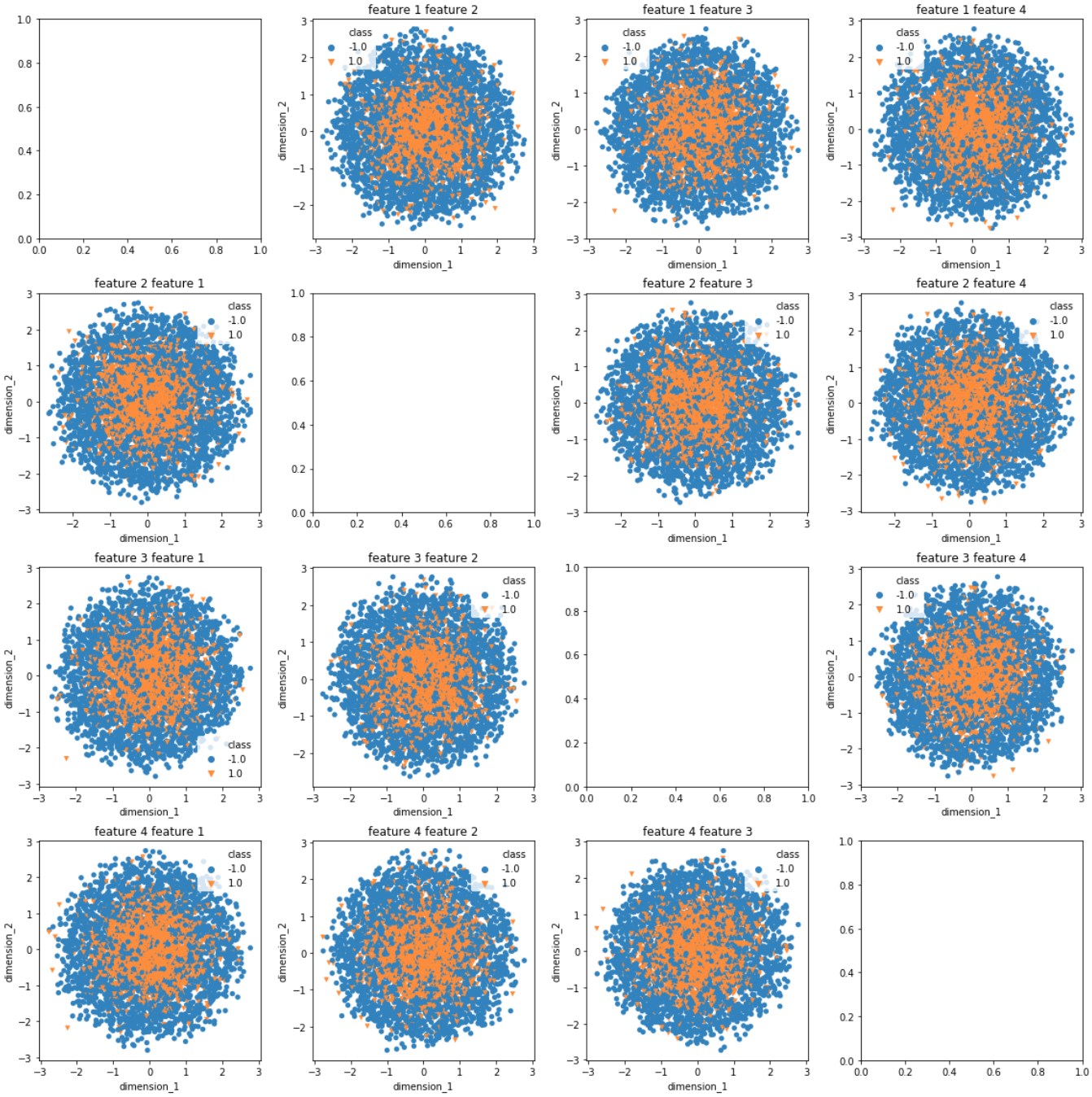
Ex 2.

1. (10 pt) Simulate training and test data as described in Section 12.3.4 of “Elements of Statistical Learning” (2nd Edition), for both the version with and the version without the 6 noise dimensions. For each version, train a SVM classifier using the RBF kernel (you can use the ’kernlab’ package in R) and tune the C and the σ^2 parameters by cross-validation.

```
Populating the interactive namespace from numpy and matplotlib
```








```
SVC(C=1.0, kernel='rbf', gamma='auto_deprecated')
```

default gamma ('auto_deprecated') = 1/feature_num

```
Out[7]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
          decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
          max_iter=-1, probability=False, random_state=None, shrinking=True,
          tol=0.001, verbose=False)
```

We use grid search to find the best hyperparameters. The two metrics are precision and recall, using either of these metrics, we can have a grid score to test the hyperparameters. The higher grid search scores mean better performance. By doing cross validation we can find the best hyperparameters

```
Best parameters set found on development set:
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
Grid scores on development set:
0.250 (+/-0.000) for {'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'}
0.250 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.501 (+/-0.423) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.940 (+/-0.011) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.954 (+/-0.012) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.250 (+/-0.000) for {'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'}
0.514 (+/-0.439) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.953 (+/-0.012) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.953 (+/-0.012) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.959 (+/-0.011) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.507 (+/-0.431) for {'C': 100, 'gamma': 1e-05, 'kernel': 'rbf'}
0.611 (+/-0.038) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.943 (+/-0.013) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.959 (+/-0.009) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.961 (+/-0.009) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.542 (+/-0.031) for {'C': 1000, 'gamma': 1e-05, 'kernel': 'rbf'}
0.953 (+/-0.011) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.954 (+/-0.012) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.960 (+/-0.010) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.956 (+/-0.014) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.
             precision    recall  f1-score   support

       -1.0       0.93      0.99      0.96       502
        1.0       0.99      0.93      0.96       498

avg / total       0.96      0.96      0.96      1000

0.96
Best parameters set found on development set:
{'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
Grid scores on development set:
0.500 (+/-0.000) for {'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.516 (+/-0.055) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.932 (+/-0.014) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.949 (+/-0.014) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'}
0.519 (+/-0.043) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.948 (+/-0.014) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.948 (+/-0.015) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.955 (+/-0.013) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.517 (+/-0.039) for {'C': 100, 'gamma': 1e-05, 'kernel': 'rbf'}
0.595 (+/-0.033) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.936 (+/-0.017) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.955 (+/-0.011) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.957 (+/-0.011) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.539 (+/-0.030) for {'C': 1000, 'gamma': 1e-05, 'kernel': 'rbf'}
0.948 (+/-0.014) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.949 (+/-0.014) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.957 (+/-0.011) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.954 (+/-0.014) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}

Detailed classification report:
```

```
The model is trained on the full development set.
The scores are computed on the full evaluation set.
             precision    recall  f1-score   support

       -1.0       0.93      0.99      0.96       502
        1.0       0.99      0.93      0.96       498

avg / total       0.96      0.96      0.96      1000

0.96
```

'C' = 100 and 'gamma' = 0.1 has best result for precision as metric 'C' = 100 and 'gamma' = 0.1 has best result for recall as metric

for orange_10 with 10 features:

```
Best parameters set found on development set:
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
Grid scores on development set:
0.250 (+/-0.000) for {'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'}
0.250 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.598 (+/-0.061) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.934 (+/-0.008) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.944 (+/-0.013) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.250 (+/-0.000) for {'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'}
0.542 (+/-0.044) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.871 (+/-0.034) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.949 (+/-0.011) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.924 (+/-0.017) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.536 (+/-0.046) for {'C': 100, 'gamma': 1e-05, 'kernel': 'rbf'}
0.580 (+/-0.053) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.938 (+/-0.010) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.956 (+/-0.009) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.906 (+/-0.015) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.532 (+/-0.034) for {'C': 1000, 'gamma': 1e-05, 'kernel': 'rbf'}
0.872 (+/-0.034) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.949 (+/-0.010) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.942 (+/-0.012) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.906 (+/-0.015) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.
             precision    recall  f1-score   support

       -1.0       0.91      1.00      0.95       502
        1.0       1.00      0.90      0.94       498

avg / total       0.95      0.95      0.95      1000

0.947
Best parameters set found on development set:
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
Grid scores on development set:
0.500 (+/-0.000) for {'C': 1, 'gamma': 1e-05, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.587 (+/-0.051) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.924 (+/-0.011) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.937 (+/-0.017) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.500 (+/-0.000) for {'C': 10, 'gamma': 1e-05, 'kernel': 'rbf'}
0.539 (+/-0.040) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.870 (+/-0.032) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.943 (+/-0.013) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.923 (+/-0.018) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.534 (+/-0.042) for {'C': 100, 'gamma': 1e-05, 'kernel': 'rbf'}
0.574 (+/-0.048) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.929 (+/-0.014) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.952 (+/-0.011) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.906 (+/-0.015) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.531 (+/-0.032) for {'C': 1000, 'gamma': 1e-05, 'kernel': 'rbf'}
0.871 (+/-0.032) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.943 (+/-0.012) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.941 (+/-0.012) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.906 (+/-0.015) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}

Detailed classification report:
```

```
The model is trained on the full development set.
The scores are computed on the full evaluation set.
             precision    recall  f1-score   support

       -1.0       0.91      1.00      0.95       502
        1.0       1.00      0.90      0.94       498

avg / total       0.95      0.95      0.95      1000

0.947
```

'C' = 100 and 'gamma' = 0.1 has best result for precision as metric 'C' = 100 and 'gamma' = 0.1 has best result for recall as metric

1. (10 pt) How does the RBF kernel perform compared to the polynomial kernels? How do you explain this performance with regard to the number of features and overfitting? (Recall that the RBF kernel is the inner product of an infinite number of features.)

The polynomial and RBF kernel are different in case of making the hyperplane decision boundary between the classes. The kernel functions are used to map the original dataset (linear/nonlinear) into a higher dimensional space with view to making it linear dataset. Polynomial kernels are less time consuming and has less ability to represent complex dataset.
The polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. In the context of regression analysis, such combinations are known as interaction features. The (implicit) feature space of a polynomial kernel is equivalent to that of polynomial regression, but without the combinatorial blowup in the number of parameters to be learned.
RBF has better feature representation ability and it can also balance the bias, variance trade off. However polynomial kernel doesn't have such ability to control overfitting.
The RBF kernel is more popular in SVM classification than the polynomial kernel. The most common degree is d=2.
We should try simplest one first and then swith to more complex ones to avoid overfitting