Machine Learning Homework1

Wenjian Hu, Jian Xu

1. Implement the k-nearest neighbor algorithm (k=1, k=5, k=50) and support vector machines with at least two different kernels with different hyperparameters. You should not simply use 'knn' or 'svm' in existing software libraries. However you are allowed to use optimization packages/libraries. For example, when implementing SVM you must formulate the optimization problem but could plug that into, say, some quadratic program solver. (20 points)

   (a) KNN Method:
   The python code is already attached, with the file name "knn.py". In the main function (main1() or main2()), we are able to choose the value of $k = 1, 5, 50$ and get corresponding results.

   Basically, we first read the training data and test data. Later, for each test instance, we search its k nearest neighbors. Then we will predict the class of the test instance according to the majority class of its neighbors.

   (b) SVM Method:
   The python code is also attached, with the name "svm.py". In the main function (main1() or main2()), we are able to choose different kernels: 'linear', 'poly' and 'rbf', and get corresponding results. In this multi-class problem, we use the one-against-all strategy.

   Basically, we need to construct k SVM models where k is the number of classes. The mth SVM is trained with all of the examples in the mth class with positive labels, and all other examples with negative labels. Thus given l training data $(x_1, y_1), ..., (x_l, y_l)$, where $x_i \in R^n, i = 1, ..., l$ and $y_i \in \{1, ..., k\}$ is the class of x_i, the mth SVM solves the following problem:

   $$min_{\omega^m, b^m, \epsilon^m} \left[\frac{1}{2}(\omega^m)^T\omega^m + C\sum_{i=1}^{l}\epsilon_i^m\right]$$
   $$\text{Subject to } y_i * [\phi(x_i)^T\omega^m + b^m] \geq 1 - \epsilon_i^m \text{ and } \epsilon_i^m \geq 0, i = 1, ..., l$$

   where the training data x_i are mapped to a higher dimensional space by the function ϕ and C is the penalty parameter.

   After solving the above problem, there are k decision functions:

   $$\phi(x)^T\omega^1 + b^1$$
   $$....$$
   $$\phi(x)^T\omega^k + b^k$$

   We say instance x is in the class which has the largest value of the decision function:

   $$\text{class of } x : argmax_{m=1,...,k}(\phi(x_i)^T\omega^m + b^m).$$

   After we build up the above quadratic problems, by using the quadratic solver (cvxopt in python), we can easily get corresponding solutions $(\omega^m, b^m, \epsilon^m)$ and then predict test data.

   Actually, the above quadratic problem is doable only for the 'linear' kernel, where $\phi(x) = x$. Generally, for other kernels, such as polynomial kernel ('poly') or radial basis kernel ('rbf'), we don't know the explicit expression of $\phi(x)$, thus we will not be able to implement the above quadratic problem. Instead, we are going to use its dual Lagrange expression:

$$maximize_{\alpha^m} \sum_i \alpha_i^m - \frac{1}{2}\sum_{i,j} \alpha_i^m \alpha_j^m y_i y_j K(x_i, x_j)$$
$$\text{where } K(x_i, x_j) = \phi(x_i)\phi(x_j)$$
$$\text{Subject to } \sum_i \alpha_i^m y_i = 0 \text{ and } C \geq \alpha_i^m \geq 0,\ i = 1, ..., l$$

This is also a quadratic problem based on α. Thus, by solving this quadratic problem, we will be able to get the value of α. Then, we can calculate the corresponding decision function:

$$f^m(x) = \phi(x)^T \omega^m + b^m$$

$$= \sum_{i=1}^{l} \alpha_i^m y_i < h(x), h(x_i) > + b^m$$

And predict the test data similarly.

2. Perform 10-fold cross validation for each algorithm and report the mean error and the confusion matrix. For the interested student (i.e. optional), compare this with the bound in the introduction to SVM learning slides for only the polynomial kernel. (10 points)

   To implement the 10-fold cross validation, the original training data is partitioned into 10 equal sized subsets according to your remainder respect to 10. Of the 10 subsets, a single subset is retained as the validation data for testing the model, and the remaining 9 subsets are used as training data. The cross-validation process is then repeated 10 times (the folds), with each of the 10 subsets used exactly once as the validation data. The 10 results from the folds can then be averaged (or otherwise combined) to produce a single estimation.

   (a) KNN 10-fold cross validation:
   For the case $k = 1$, the overall accuracy is: 99.27921568627451%
   For the case $k = 1$, the combined confusion matrix is:

$$\begin{bmatrix} & P0 & p1 & P2 & p3 & P4 & p5 & P6 & p7 & P8 & p9 \\ A0 & 382 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ A1 & 0 & 386 & 2 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ A2 & 0 & 2 & 389 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ A3 & 0 & 0 & 2 & 365 & 0 & 1 & 0 & 2 & 0 & 0 \\ A4 & 0 & 0 & 0 & 0 & 389 & 0 & 0 & 1 & 0 & 1 \\ A5 & 0 & 0 & 0 & 0 & 0 & 374 & 0 & 0 & 0 & 1 \\ A6 & 0 & 0 & 0 & 0 & 0 & 0 & 351 & 0 & 0 & 0 \\ A7 & 0 & 2 & 1 & 1 & 0 & 0 & 0 & 370 & 0 & 0 \\ A8 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 361 & 0 \\ A9 & 0 & 1 & 0 & 0 & 2 & 1 & 0 & 0 & 0 & 352 \end{bmatrix}$$

   In the conbined confusion matrix, $P0, P1, ..., P9$ mean predict $0, 1, ..., 9$ and $A0, A1, ..., A9$ mean Actual $0, 1, ..., 9$.

   For the case $k = 5$, we get similar results, but with a slightly smaller overall accuracy: 98.95864527629233%. Most nonzero terms in the confusion matrix also appear in the diagonal.

   For the case $k = 50$, we also get similar results, but with a smaller overall accuracy: 93.37953654188951%. Most nonzero values also appear in the diagonal terms. The reason why $k = 50$ gives less accuracy is that most of the data with the same class are very close

to each other. Thus, if $k = 1$ or $k = 5$, the test instance can easily find its nearest neighbors and the correct class. If $k = 50$, some nearest neighbors can belong to other classes, which might lead to miss-prediction.

The detailed 10-fold cross validation results are also attached, with file names "knn confusion matrix k=1", "knn confusion matrix k=5" and "knn confusion matrix k=50".

(b) SVM 10-fold cross validation:
For the kernel 'linear', the overall accuracy is: 94.76705882352941%
For the kernel 'linear', the combined confusion matrix is:

$$\begin{bmatrix} & P0 & p1 & P2 & p3 & P4 & p5 & P6 & p7 & P8 & p9 \\ A0 & 360 & 2 & 0 & 0 & 3 & 1 & 1 & 1 & 16 & 0 \\ A1 & 0 & 354 & 9 & 7 & 0 & 11 & 1 & 2 & 0 & 6 \\ A2 & 0 & 6 & 386 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A3 & 0 & 3 & 1 & 364 & 0 & 0 & 0 & 2 & 0 & 0 \\ A4 & 0 & 1 & 0 & 0 & 383 & 3 & 0 & 2 & 0 & 2 \\ A5 & 0 & 1 & 0 & 7 & 1 & 334 & 2 & 2 & 2 & 26 \\ A6 & 0 & 0 & 0 & 0 & 1 & 0 & 348 & 0 & 2 & 0 \\ A7 & 0 & 0 & 0 & 1 & 1 & 3 & 0 & 364 & 5 & 0 \\ A8 & 13 & 3 & 0 & 0 & 0 & 12 & 0 & 1 & 333 & 1 \\ A9 & 2 & 8 & 0 & 4 & 0 & 8 & 0 & 0 & 10 & 324 \end{bmatrix}$$

In the conbined confusion matrix, $P0, P1, ..., P9$ mean predict $0, 1, ..., 9$ and $A0, A1, ..., A9$ mean Actual $0, 1, ..., 9$.

For the kernel 'poly', we get similar results, but with a better overall accuracy: 98.82538324420676%. Most nonzero terms in the confusion matrix also appear in the diagonal.

For the kernel 'rbf', we get very bad predictions, with an overall accuracy: 10.411122994652406%

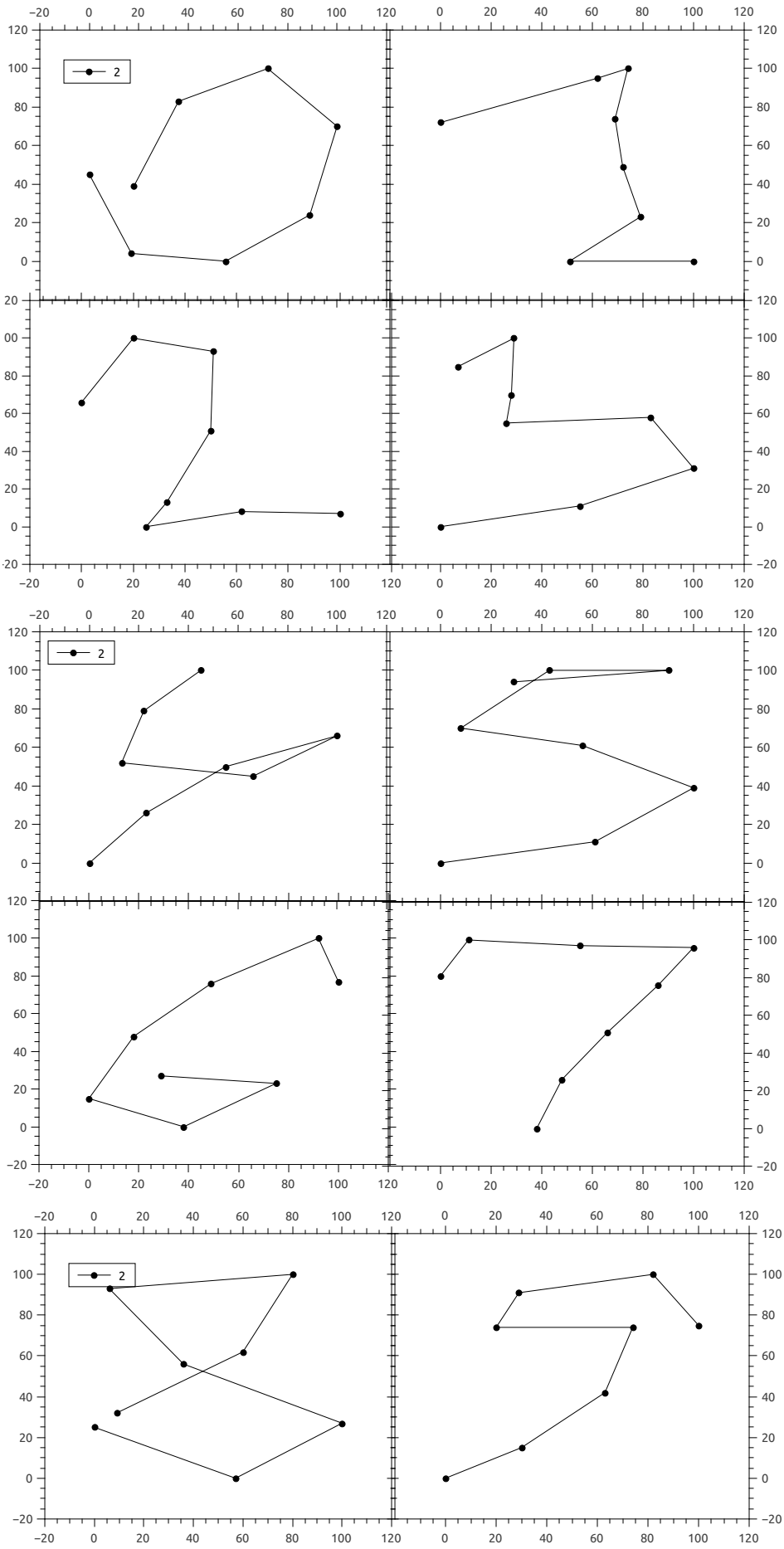
The detailed 10-fold cross validation results are also contained in the file names "svm confusion matrix linear", "svm confusion matrix poly" and "svm confusion matrix rbf".

3. For the SVM formulations draw an example instance corresponding to the support vector for each digit (10 points).

The support vector for the mth digit here is x_i which satisfy $f^m(x_i) = \phi^T(x_i)\omega^m + b^m = +1$, where $f^m(x)$ is the mth solved decision function from previous quadratic problem.

For the linear kernel, I have found corresponding support vectors, which are contained in the file name "support vectors svm linear".

To draw an example instance for each digit, we have:


2
2
2

4. Attempt to explain the difference in overall errors between the kNN algorithm and SVM algorithm. Consider what assumptions the underlying algorithms are making. Formulate your explanation as a hypothesis and construct a simple experiment to verify it. (10 points)

   KNN algorithm is a method for classifying objects based on closest training examples, which is among the simplest of all machine learning algorithms. A main advantage of the KNN algorithm is that it performs well with multi-modal classes because the basis of its decision is based on a small neighborhood of similar objects. And from the 10-fold cross validation, we do see a very good overall accuracy for $k = 1$ and $k = 5$. The accuracy for $k = 50$ goes down, because there are too many neighbors and some nearest neighbors can belong to other classes. A major disadvantage of the KNN algorithm is that it uses all the features equally in computing for similarities. This can lead to classification errors, especially when there is only a small subset of features that are useful for classification.

   SVM classification uses different planes in space to divide data points. A main advantage of SVM classification is that SVM performs well on datasets that have many attributes, even when there are only a few cases that are available for the training process. However, several disadvantages of SVM classification include limitations in speed and the selection of the kernel functions. For example, from the 10-fold cross validation, we see 'linear' kernel and 'poly' kernel both give very good predictions but 'rbf' kernel gives very bad predictions. Thus, we should be very careful about the selection of kernel functions.

   SVM assumes there exist a hyperplane separating the data points which is a quite restrictive assumption (global consistency), while kNN attempts to approximate the underlying distribution of the data in a non-parametric fashion, basically assuming data belonging to the same class belong to the same cluster (local consistency).

5. Predict the digit being written for each instance in the the test data set we make available. One prediction per line in the order the data is given. (10 points)

   The "KNN" prediction (k=5) is attached, with the file name "predict test data knn k=5".

   The "SVM" prediction (kernel='poly') is attached, with the file name "predict test data svm poly".

6. Implement the transfer SVM. Train your transfer SVM on the source problem (0vs8Source.csv) of differentiating 0s from 8s for one set of writers (1-20) and transfer transfer what was learnt to differentiate 0s from 8s for another set of writers (21-30) (0vs8Target.csv) . Report your results on the test data set (0vs8NoLabelTest.csv) (20 points)

   To implement the transfer SVM, we are going to solve the following quadratic problem (main3() or main4() in svm.py):

$$min_{\omega,b,\epsilon} \left[\frac{1}{2}(\omega)^T\omega + C\sum_{i=1}^{l}\epsilon_i\right]$$
$$\text{Subject to } y_i * f^{source}(x_i) + y_i * [\phi(x_i)^T\omega + b] \geq 1 - \epsilon_i \text{ and } \epsilon_i \geq 0, i = 1, ..., l$$

   Where $f^{source}(x)$ is the decision function formulated from the source data (0vs8Source.csv). And $[\phi(x_i)^T\omega + b]$ is the decision function will be formulated from the target data (0vs8Target.csv).

After get these two decision function, we can formulate the final decision function:

$$F(x) = f^{source}(x) + [\phi(x)^T \omega + b]$$

If $F(x) > 0$, then it's predicted as 0 (8), else it's predicted as 8 (0). To estimate how accurately my transfer SVM model will perform in practice, I also do the 10 fold cross validation and here is the result:

The overall accuracy is: 92.682927%
The combined confusion matrix is:

$$\begin{bmatrix} & P0 & p8 \\ A0 & 20 & 1 \\ A8 & 2 & 18 \end{bmatrix}$$

In the conbined confusion matrix, $P0, P1, ..., P9$ mean predict $0, 1, ..., 9$ and $A0, A1, ..., A9$ mean Actual $0, 1, ..., 9$. So from the confusion matrix, it seems the transfer SVM is good.

The transfer SVM prediction about the test data set (0vs8NoLabelTest.csv) is also attached, with the file name "transfer svm predicts test data".