IDC Machine Learning From Data HW 5: SVM and Lagrange Multipliers

In this HW we will use Weka's implementation of an SVM solver/classifier to predict election results as well as answer some theoretical questions regarding kernel functions and Lagrange multipliers.

You will also be implementing the wrapper method for feature selection.

This Hw is to be done in pairs.

A note on programming assignments in this course: You are allowed to use the current version of WEKA and the current version of Java for programming assignments in this course. If your code does not compile using these two packages, you will lose 10 points for not having code which compiles. If you are using an old version of java it is recommended that you update your version and test compilation with the new version.

Note: All cross validation estimates of the error should use 3 folds.

Instructions for the programming assignment.

SVM in WEKA:

1. Download the hw5.zip folder
2. Inside of a project (e.g. Homework4) create a package called hw4
3. Move SVMEval.java into this package
4. Split the elections data file into two files. One for validation and for testing. The validation file should contain 200 instances. The test should contain the rest. You will use the validation file for feature selection and optimization of parameters. You will the test file for reporting your final error. You can do this with your favorite text editor. Name the training file `ElectionsData_train.txt` and the test file `ElectionsData_test.txt.` These files should be places in the hw4 package folder so that we do not have to worry about absolute path file names.
5. In the constructor method of SVMEval.java you should instantiate an SVM classifier.
6. Implement a method **buildClassifier** which receives as input instances object and builds the SVM classifier. This method should simply call the buildClassifier method of the WEKA SVM classifier that we are using: mySmoObject.buildClassifier(instances).
7. Implement a method named **chooseKernel** which chooses the best kernel.

   Input: Instances

   Action: chooses best kernel from the options: 13 possible RBF kernels with parameter gamma 2^i for -10<=i<=2, 3 possible polynomial kernels of degree i for 2<=i<=4. The chosen kernel is the kernel that yields the best cross validation error for the SVM model that uses it.

   Output: the method sets the kernel of your SVM classifier.
8. Implement a method **backwardsWrapper** which performs feature selection (explanation after programming instructions):

   Input: threshold, min number of attributes, instances

Action: performs the backwards wrapper feature selection algorithm as explained above.
Output: New Instances object with the chosen subset of the original features, indices of chosen features.
Note: After using this method, note that if you use other Instances objects and you want to use the subset of features you should remove the features that backwardsWrapper chose to remove.

9. Implement a method called **calcAvgError** :

   Input: Instances.

   Output: The number of prediction mistakes the classifier makes divided by the number of instances.

10. Implement a method called **calcCrossValidationError**:

    Input: Instances object.

    Output: The cross validation estimate of the avg error.

11. In your main: load the data

    On the validation set:

    1. Choose best kernel using the chooseKernel method.

    2. Perform features selection using backwardWrapper. **You should use a threshold of 0.05 and a minNumberOfFeatures of 5.**

    3. Build the classifier

    On the test set:

    1. Leave only selected features

    2. Calculate average error using the trained classifier.

    3. Print the average error.

    See an example for a main method below.

    NOTE: For all cross validation estimates use 3 folds

Explanation of The Wrapper Method:

Intro: Feature selection

Feature selection is the process of choosing a subset of the original features. Usually you do this because there are too many features to process or many irrelevant features. When you choose a subset of the features you want to choose the features that help you classify the most.

The wrapper method is a greedy algorithm for feature selection. There is a backwards and a forwards version.

The backwards version (with input k, and threshold t):

```
error_diff = 0;

original_error = crossvalidation error of SVM using all features;

S = feature set;

Do this until |S| = k or error_diff > threshold: {

        i_minimal = 1;

         minimal_error = the SVM cross validation error on the set of features S without {x_1}

        For each i in {2…|S|} {

                    new_error = the SVM cross validation error on the set of features S without {x_i}

                    if (new_error < minimal_error) {

                                minimal_error = new_error;

                                 i_minimal = i;

                    }

          }

          error_diff = minimal_error – original_error

          If (error_diff < threshold  t){

                    Remove X_i_minimal from S

          }

}
```

Things For WEKA:

<u>Using WEKA's SVM classifier:</u>

Creating an SVM classifier:

```
SMO smo = new SMO();
PolyKernel kernel = new PolyKernel();
smo.setKernel(kernel);
smo.buildClassifier(data);
```

Here a polynomial kernel is used but you can choose a different kernel. For Example The RBF kernel is created like so:

```
RBFKernel kernel = new RBFKernel();
```

Then in order to make a prediction on a new instance you just:

```
smo.classifyInstance(someInstance);
```

How to remove attributes from instances object:

To remove an attribute:

```
Remove remove = new Remove();
remove.setInputFormat(curInstances);
String[] options = new String[2];
options[0] = "-R";
options[1] = Integer.toString(someIndex + 1);
remove.setAttributeIndicesArray(options);
Instances workingSet = Filter.useFilter(curInstances, remove);

//Where Remove comes from weka.filters.unsupervised.attribute.Remove;
```

Setting Kernel parameters:

To set the degree of a polynomial kernel:

```
PolyKernel kernel = new PolyKernel();
kernel.setExponent(degree);
```

To set the gamma parameter of an RBF kernel:

```
RBFKernel kernel2 = new RBFKernel();
kernel2.setGamma(0.3);
```

Example for a main method

Your method signatures can be different, so the input to these methods might be different:

```
String training = "ElectionsData_train.txt";
String testing = "ElectionsData_test.txt";
BufferedReader datafile = readDataFile(training);

Instances data = new Instances(datafile);
```

```
            data.setClassIndex(0);
            SvmEval eval = new SvmEval();
            eval.chooseKernel(data);
            Instances workingSet = eval.backwardsWrapper(data, 0.05, 5);

            eval.buildClassifier(workingSet);

            BufferedReader datafile2 = readDataFile(testing);
            Instances dataTest = new Instances(datafile2);
            dataTest.setClassIndex(0);
            Instances subsetOfFeatures =
eval.removeNonSelectedFeatures(dataTest);

            double avgError = eval.calcAvgError(subsetOfFeatures);

            System.out.println(avgError);
```

Theoretical Questions:

1. Show that any Kernel function is symmetric, i.e. $K(x_i, x_j) = K(x_j, x_i)$
1. Define K to be the radial basis kernel function: $K(x_i, x_j) = \exp\left(-\frac{1}{2}\|x_i - x_j\|^2\right)$ for every $x_i, x_j \in \mathbb{R}^n$. The feature space of this kernel has infinite number of dimension, thus there is a mapping: $\phi(x)$: $\mathbb{R}^n \to \mathbb{R}^\infty$ such that $\langle \phi(x_j), \phi(x_i) \rangle = K(x_j, x_i)$.
   Show that for any two input instances $x_i, x_j \in \mathbb{R}^n$, $\|\phi(x_i) - \phi(x_j)\|^2 \leq 2$

Note: Here we can see that we have a kernel function and some implicit mapping, which we do not know and is infinite dimensional and we can still prove things about points in the feature space. That is we can still make claims about any mapped point's distance from each other without knowing what the mapping or the space is.

2. Use Lagrange Multipliers to find the maximum and minimum values of the function subject to the given constraints:

   a.) $f(x, y) = 3e^{xy}$; constraint: $x^2 + y^2 = 32$

   b.) $f(x, y) = 3\pi x^2 y$; constraint $6 + 6\pi xy + 3\pi x^2 = 12$

   c.) $f(x, y, z) = 3x - y - 3z$; constraints: $x + y - z = 0$; $x^2 + 2z^2 = 1$