**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ**

**Кафедра информационных систем**

**ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ  
по курсу: Теория информационных процессов и систем**

**Отбор признаков**

**Работу выполнили студенты:**

**Трофимов Владислав**

**Виноградов Павел**

**Группа 3511**

**САНКТ-ПЕТЕРБУРГ**

**2015**

В качестве исходных данных для лабораторной работы была взята статистика по лесным пожарам. Критериями являются 13 различных параметров, в их числе влажность, близость водоёмов, месяц, напрвление ветра и другие. Все критерии имеют числовой тип или представляются в числовом типе, те легко их легко изучать. Результирующим критерием является отсутствие (0) или присутствие (1) возгарания. Данная статистика имеет 515 экземпляров.

Список имеющихся атрибутов:

* @ATTRIBUTE X – координата Х внутри парка
* @ATTRIBUTE Y - координата У внутри парка
* @ATTRIBUTE month - месяц
* @ATTRIBUTE day - день
* @ATTRIBUTE FFMC - торфяной слой
* @ATTRIBUTE DMC - влажность
* @ATTRIBUTE DC - влажность нижних слоев
* @ATTRIBUTE ISI - предполагаемая скорость распространения
* @ATTRIBUTE temp - температура
* @ATTRIBUTE RH - влажность
* @ATTRIBUTE wind - ветер
* @ATTRIBUTE rain - дождь
* @ATTRIBUTE fired - наличие возгарания

Внешним критерием для оценки был выбран критерий скользящего контроля.

В качестве первого алгоритма отбора было выбрано последовательное добавление признаков. Мы выбираем тот признак, при использовании которого в контрольной выборке получается наибольшее количество правильных предсказаний. Останавливается алгоритм тогда, когда это количество начинает уменьшаться.

Алгоритм отбора признаков был написан на языке Java с применение библиотеки Weka.

Исходный код:

**package** com.ifmo.year2015.group3511.trofimov;  
  
**import** weka.classifiers.Evaluation;  
**import** weka.classifiers.trees.J48graft;  
**import** weka.core.Instances;  
**import** weka.filters.Filter;  
**import** weka.filters.unsupervised.attribute.Remove;  
  
**import** java.io.BufferedReader;  
**import** java.io.FileNotFoundException;  
**import** java.io.FileReader;  
**import** java.io.IOException;  
**import** java.util.\*;  
  
*/\*\*  
 \** ***@author*** *vladislav.trofimov@emc.com  
 \*/***public class** Main {  
  
 **private static final** String ***INPUT\_FILE\_NAME*** = **"data.arff"**;  
  
 **public static void** main(**final** String[] args) **throws** Exception {  
 Instances data = **null**;  
 **try** {  
 BufferedReader reader = **new** BufferedReader(**new** FileReader(***INPUT\_FILE\_NAME***));  
 data = **new** Instances(reader);  
 reader.close();  
 } **catch** (**final** FileNotFoundException e) {  
 System.***out***.println(**"Can't find file "** + ***INPUT\_FILE\_NAME***);  
 } **catch** (**final** IOException e) {  
 System.***out***.println(**"Error parsing file "** + ***INPUT\_FILE\_NAME***);  
 }  
 **if** (data != **null**) {  
 data.setClassIndex(data.numAttributes() - 1);  
 *processSequentialAttributeAdditionAlgorithm*(data);  
 *processBruteForceAlgorithm*(data);  
 }  
 }  
  
 **private static void** processBruteForceAlgorithm(**final** Instances data) {  
 **if** (data.numAttributes() > 63) {  
 **throw new** IllegalArgumentException(**"Trying to brute-force more than 63 attributes"**);  
 }  
 **long** noAttributesValue = 1 << data.numAttributes() - 1;  
 **long** allAttributesValue = (1 << data.numAttributes()) - 1;  
 **long** totalAttributeSetCount = allAttributesValue - noAttributesValue;  
 **try** {  
 System.***out***.println(**"Starting brute-force algorithm"**);  
 **long** bestAttributeSet = 0;  
 **double** bestAttributeSetQuality = Double.***MIN\_VALUE***;  
 **for** (**long** currentAttributeSet = noAttributesValue + 1; currentAttributeSet <= allAttributesValue; currentAttributeSet++) {  
 **double** attributeSetQuality = *getQualityForAttributeSet*(*getAttributesOptions*(Long.*toBinaryString*(currentAttributeSet)), data);  
 **if** (attributeSetQuality > bestAttributeSetQuality) {  
 bestAttributeSetQuality = attributeSetQuality;  
 bestAttributeSet = currentAttributeSet;  
 }  
 *printProgress*(currentAttributeSet - noAttributesValue, totalAttributeSetCount, bestAttributeSetQuality);  
 }  
 System.***out***.println();  
 *printAttributeSet*(data, Long.*toBinaryString*(bestAttributeSet), bestAttributeSetQuality);  
 } **catch** (**final** Exception e) {  
 System.***out***.println(**"Incorrect decision tree usage"**);  
 e.printStackTrace();  
 }  
 }  
  
 **private static void** processSequentialAttributeAdditionAlgorithm(**final** Instances data) {  
 LinkedList<Integer> selectedAttributesIndexes = **new** LinkedList<>();  
 LinkedList<Double> selectedAttributesQuality = **new** LinkedList<>();  
 ArrayList<Double> iteratingAttributesQuality = **new** ArrayList<>();  
 ArrayList<Integer> iteratingAttributesIndexes = **new** ArrayList<>();  
 **try** {  
 System.***out***.println(**"Starting sequential attribute adding algorithm"**);  
 **while** (**true**) {  
 iteratingAttributesQuality.clear();  
 iteratingAttributesIndexes.clear();  
 **for** (**int** currentAttribute = 1; currentAttribute < data.numAttributes(); currentAttribute++) {  
 String[] selectedAttributesOptions = *getOptionsForFilteredAttributes*(data.numAttributes(), currentAttribute, selectedAttributesIndexes);  
 **if** (selectedAttributesOptions == **null**) {  
 **continue**;  
 }  
 **double** attributeSetQuality = *getQualityForAttributeSet*(selectedAttributesOptions, data);  
 iteratingAttributesQuality.add(attributeSetQuality);  
 iteratingAttributesIndexes.add(currentAttribute);  
 }  
 **if** (iteratingAttributesQuality.isEmpty()) {  
 **break**;  
 }  
 **int** bestQualityAttributeIndex = iteratingAttributesQuality.indexOf(Collections.*max*(iteratingAttributesQuality));  
 **if** (*qualityDecreased*(iteratingAttributesQuality, selectedAttributesQuality, bestQualityAttributeIndex)) {  
 **break**;  
 }  
 **double** addedAttributeQuality = iteratingAttributesQuality.get(bestQualityAttributeIndex);  
 **int** addedAttributeIndex = iteratingAttributesIndexes.get(bestQualityAttributeIndex);  
 System.***out***.println(data.attribute(addedAttributeIndex - 1) + **" added with total set quality "** + addedAttributeQuality);  
 selectedAttributesQuality.add(addedAttributeQuality);  
 selectedAttributesIndexes.add(addedAttributeIndex);  
 }  
 System.***out***.println();  
 } **catch** (**final** Exception e) {  
 System.***out***.println(**"Incorrect decision tree usage"**);  
 e.printStackTrace();  
 }  
 }  
  
 **private static** String[] getOptionsForFilteredAttributes(**final int** classIndex, **final int** currentAttribute, **final** List<Integer> attributes) {  
 **if** (attributes.contains(currentAttribute)) {  
 **return null**;  
 }  
 String[] options = **new** String[3];  
 options[0] = **"-R"**;  
 StringBuilder stringBuilder = **new** StringBuilder();  
 stringBuilder.append(classIndex);  
 stringBuilder.append(**','**);  
 stringBuilder.append(currentAttribute);  
 **for** (**final int** attribute : attributes) {  
 stringBuilder.append(**','**);  
 stringBuilder.append(attribute);  
 }  
 options[1] = stringBuilder.toString();  
 options[2] = **"-V"**;  
 **return** options;  
 }  
  
 **private static boolean** qualityDecreased(**final** List<Double> iteratingAttributesQuality, **final** LinkedList<Double> selectedAttributesQuality, **final int** bestQualityAttributeIndex) {  
 **return** !selectedAttributesQuality.isEmpty() && iteratingAttributesQuality.get(bestQualityAttributeIndex) < selectedAttributesQuality.getLast();  
 }  
  
  
 **private static void** printProgress(**final double** current, **final double** total, **final double** bestValue) {  
 System.***out***.printf(Locale.***ENGLISH***, **"\rCalculation progress: %10.7f with best attribute set quality: %5.2f"**, current / total \* 100.0, bestValue);  
 }  
  
 **private static** String[] getAttributesOptions(**final** String binaryInterpretation) {  
 String[] options = **new** String[3];  
 options[0] = **"-R"**;  
 StringBuilder stringBuilder = **new** StringBuilder();  
 **for** (**int** attributeIndex = binaryInterpretation.length() - 1; attributeIndex >= 0; attributeIndex--) {  
 **char** attribute = binaryInterpretation.charAt(binaryInterpretation.length() - attributeIndex - 1);  
 **if** (attribute == **'1'**) {  
 stringBuilder.append(**','**);  
 stringBuilder.append(attributeIndex + 1);  
 }  
 }  
 options[1] = stringBuilder.substring(1);  
 options[2] = **"-V"**;  
 **return** options;  
 }  
  
 **private static void** printAttributeSet(**final** Instances data, **final** String binaryInterpretation, **final double** quality) {  
 **for** (**int** attributeIndex = 0; attributeIndex < binaryInterpretation.length() - 1; attributeIndex++) {  
 **char** attribute = binaryInterpretation.charAt(binaryInterpretation.length() - attributeIndex - 1);  
 **if** (attribute == **'1'**) {  
 System.***out***.println(data.attribute(attributeIndex));  
 }  
 }  
 System.***out***.println(**"Quality: "** + quality + **'\n'**);  
 }  
  
 **private static double** getQualityForAttributeSet(**final** String[] selectedAttributesOptions, **final** Instances data) **throws** Exception {  
 Remove remove = **new** Remove();  
 remove.setOptions(selectedAttributesOptions);  
 remove.setInputFormat(data);  
 Instances newData = Filter.*useFilter*(data, remove);  
 String[] options = **new** String[1];  
 options[0] = **"-U"**;  
 J48graft tree = **new** J48graft();  
 tree.setOptions(options);  
 tree.buildClassifier(newData);  
 Evaluation eval = **new** Evaluation(newData);  
 eval.crossValidateModel(tree, newData, 10, **new** Random(1));  
 **return** eval.correct();  
 }  
  
}

Результаты первого алгоритма:

Starting sequential attribute adding algorithm

@attribute Y numeric added with total set quality 284.0

@attribute X numeric added with total set quality 299.0

@attribute temp numeric added with total set quality 307.0

@attribute ISI numeric added with total set quality 313.0

@attribute wind numeric added with total set quality 313.0

@attribute rain numeric added with total set quality 313.0

Результаты второго алгоритма:

Starting brute-force algorithm

Calculation progress: 100.0000000 with best attribute set quality: 313.00

@attribute X numeric

@attribute Y numeric

@attribute ISI numeric

@attribute temp numeric

Quality: 313.0

Так как атрибут «дождь» не приводит к увеличению значения внешнего критерия, он является нерелевантным. Исходя из этого можно придти к выводу, что алгоритм полного перебора производит выборку необходимых атрибутов лучше нежели алгоритм последовательного добавления признаков.

По результатам выборки мы видим, что наиболее важными атрибутами являются координаты возникновения возгорания относительно центра леса, а также предполагаемая скорость распространения и температура. Так как предполагаемая скорость распространения является параметром линейно зависящим от торфяного слоя и ветра, необходимость в этих атрибутах (торфяной слой и ветер) отпадает.