**Assignment 5 Report**

# Data Description

**DataSet: Crowdsourced Mapping (**[**http://archive.ics.uci.edu/ml/datasets/Crowdsourced+Mapping**](http://archive.ics.uci.edu/ml/datasets/Crowdsourced+Mapping)**)**

Number of instances in dataset: 10546

Number of attributes in dataset: 29

Default Task: Classification

Data Types: Multivariate

Year: 2016

Output Class: Different land cover classes (impervious, farm, forest, grass, orchard, water)

# Pre-Processing

We did not find any *missing/NULL/Nan* values in the dataset, so we did not think it required any padding of nearby values. The dataset contained negative values, and when run with *Multinomial Naïve Bayes* classifier from *sklearn*, we realized that *Multinomial Naïve Bayes* does not take negative values. Hence, a scaling of values was done scaling them to the range (0, 1) and all the algorithms were run after this scaling. To select best parameters, *SelectKBest* from *sklearn* was used. We found that we were getting a good accuracy when *k* was equal to *20*, which means that 9 of the attributes either did not contribute to the accuracy were correlated to some other attribute. Other than this, no other sort of pre-processing was performed.

# Finding Best Classifier Parameters

We run every classifier using methods from *scikit-learn* in *Python* programming language and attempt to change the important and known parameters so that it makes some difference in the accuracy. At each change we are hoping for a better accuracy. Each of the algorithms were run a couple of times depending on the number of important parameters in the function call and the different values those parameters could take. The Logs.docx file in the main folder gives you a list of classifiers with their changing parameters and the effect of those changes on the accuracy. The ones giving the best performance out of the parameter values tested have been highlighted (in bold) in each of the log tables.

# Testing All Classifiers Together

All classifiers together were run using the pseudocode as below:

numFolds = k (where k >= 10)

classifiers = {c1, c2, …, cn} // list of n classifiers with best parameters

split the data into k folds d[1…k]

for i in 1 to numFolds :

// create training dataset by combining all folds except d[i]

train = {d[1] + d[2] + … + d[i-1] + d[i+1] + … + d[k]}

// create test dataset using d[i]

test = d[i]

for c in classifiers:

// create a model of type c using train

model <- createModel(c, train)

// find accuracy of model of type c on test

for classifier c: accuracy[i] <- findAccuracy(model, test)

for classifier c: other\_parameter[i] <- findEvaluation(model, test)

next c

next i

# Results

Number of instances in dataset: 10546

Number of attributes in dataset: 20 (selected after pre-processing, else 29)

How many fold cross-validation performed: 10-fold

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Classifier** | **Best Parameters Used** | **Accuracy** | **F1 score** | **Precision** |
| Decision Tree | Max-depth = 20  Min-Sample size = 25  Max leaf Nodes = 100 | 0.85473283323 | 0.8616086816996 | 0.8598153584801 |
| Perceptron | Penalty=l1  Alpha=0.0001 | 0.761320323985 | 0.7382619048150 | 0.8080752820486 |
| Neural Network | Hidden layer sizes=(30,15)  Learning Rate=invscaling | 0.884613517772 | 0.8719745961784 | 0.8825316487523 |
| SVM | Kernel=linear  C=0.4 | 0.866689409431 | 0.8556934895382 | 0.8563033311528 |
| KNN | No. of neighbours = 5  Power parameter  = 2  leaf size = 30 | 0.900728253489 | 0.8987155293820 | 0.9103169844055 |
| Random forest | No. of Estimators = 150  Max Depth = None  Max leaf nodes = None | 0.910869349236 | 0.90393819531 | 0.9025578036571 |
| Bagging | No. of Estimators = 100  Max features = 0.5  Bootstrap = False | 0.915239059593 | 0.906253092101 | 0.9057498018608 |
| Adaboost | No. of Estimators = 100  Learning rate = 0.2 | 0.842014482419 | 0.8354813802439 | 0.8416728981593 |
| Logistic Regression | Penalty = l1  C = 2 | 0.859769412081 | 0.8397697812807 | 0.8432029319193 |
| Gradient Boosting | No. of Estimators = 200  Learning rate = 0.1  Max Depth = 4 | 0.908409450281 | 0.9094799631254 | 0.9108579413826 |
| Deep Learning | Hidden Layer Sizes = (40,30,20,10,5,2,1  Learning rate = Invscaling | 0.7609506614294 | 0.6796383344965 | 0.6232169396766 |
| Naïve Bayes | Alpha = 0.1 | 0.704695785337 | 0.582621835473 | 0.4965971490056 |

# Analysis

We attempted to use more than just one extra evaluation metric to find a pattern of sorts and see if using accuracy was the best option or we should go with some other evaluation metric. We noticed some similarities and some places where accuracy and the other metrics, f1-score and precision in our case, to be different.

As we can see from the table in the results, there are places where all the three metrics give very similar outputs. These were very prominent in almost all the results having accuracy greater than 80% or 0.80. So, it seemed that the precision and f1-score measures were no good as compared to accuracy. Accuracy metric is the best of them all when it did not drop 80%. As it drops from 80%, especially in the case of Naïve Bayes and Deep Learning, you can see a steep reduction in the precision as well as the f1-score. Precision addresses the number of true positives and reduces when the true positives in the output are low. Which means that these two classifiers have lesser true positives as compared to others. The f1-score is based on both precision and recall, hence the true positives are reducing with the number of relevant instances classified correctly. This means that the classifier is performing poorly. As an exception to this, perceptron classifier even though gives a bad accuracy, actually has a higher number of true positives.

From the experience gained on this assignment, I think we cannot just rely on one single evaluation metric and should use more than one metric to identify problems in our classifiers.

Naïve Bayes, Deep Neural Network and Perceptron seem to be the three weak classifiers over the dataset. Naïve Bayes did not work well because the dataset contains a lot of noise and most of the attributes seemed correlated to each other, thus the naïve assumption that all attributes are independent did not work well. Deep Neural Network and Perceptron may have been a problem due to the fact that the data was scaled between the range (0, 1) which meant that the dataset after scaling had very small changes in values, which may not have converged in the default 1000 max iterations.

Some of these classifiers were slow in performance whereas some others were very fast. KNN performed the fastest computation, probably because of the fact that it is a lazy learner and it only takes one pass through the data. Others take multiple passes through the data to learn. Deep Learning was mostly the slowest in performance, and in this case not very good classifier as well, due to the fact that the number of parameters in a deep learning framework increases with an increase in the number of layers.