Assignment: Multi-class Classification

INF131: Data Science and Web Mining

Giannis Nikolentzos[1] and Michalis Vazirgiannis[1,2]

[1]*Department of Informatics , Athens University of Economics and Business*
[2]*LIX , Ecole Polytechnique*

November 2015

1 Description of the Assignment

In this assignment, you will deal with a classification problem from Kaggle[1]. Kaggle is a platform for predictive modelling and analytics competitions on which companies and researchers post their data and statisticians and data miners from all over the world compete to produce the best models. This allows companies and researchers to investigate which method is most effective for addressing a given problem. To encourage more people to participate in a competition, often, companies award prizes to the highest ranked participants.

As part of this assignment, you will participate in the competition organized by Walmart[2], an American multinational retail corporation that operates a chain of hypermarkets, discount department stores and grocery stores. Your task is to build a classification system that predicts shopping trip types based on the items that customers purchased. For example, a customer may make a small daily dinner trip, a weekly large grocery trip, a trip to buy gifts for an upcoming holiday, or a seasonal trip to buy clothes. Walmart has categorized the trips contained in this data into 38 distinct types using an extended dataset. Your goal is to recreate this categorization with a more limited set of features. Predicting trip types more accurately will allow Walmart to provide improved shopping experiences to customers.

2 Dataset

You are given the `train.csv` file which contains a large number of customer visits. Each line in the file corresponds to an item that was purchased by a customer and consists of the following fields:

1. **TripType**: A categorical id representing the type of shopping trip the customer made. This is the ground truth that you are predicting.
2. **VisitNumber**: An id corresponding to a single trip by a single customer. Note that the all products bought by a customer during a single trip will have the same *VisitNumber*.
3. **Weekday**: The weekday of the trip. Products bought by a customer in a single session will have same *Weekday* values.
4. **Upc**: The UPC number of the product purchased.

[1] `http://www.kaggle.com`
[2] `http://www.kaggle.com/c/walmart-recruiting-trip-type-classification`

5. **ScanCount**: The number of the given item that was purchased. A negative value indicates a product return.

6. **DepartmentDescription**: A high-level description of the item's department.

7. **FinelineNumber**: A more refined category for each of the products, created by Walmart.

For example, below are given the lines corresponding to the first 6 items.

TripType	VisitNumber	Weekday	Upc	ScanCount	DepartmentDescription	FinelineNumber
999	5	Friday	68113152929	-1	FINANCIAL SERVICES	1000
30	7	Friday	60538815980	1	SHOES	8931
30	7	Friday	7410811099	1	PERSONAL CARE	4504
26	8	Friday	2238403510	2	PAINT AND ACCESSORIES	3565
26	8	Friday	2006613744	2	PAINT AND ACCESSORIES	1017
26	8	Friday	2006618783	2	PAINT AND ACCESSORIES	1017
26	8	Friday	2006613743	1	PAINT AND ACCESSORIES	1017

As mentioned earlier, a specific *VisitNumber* corresponds to a single visit of a customer. For example, for the visit with number 7, the customer bought two products belonging to departments *SHOES* and *PERSONAL CARE* respectively. Moreover, since both products were purchased at the same visit, their *Weekday* and *TripType* values are the same (*Weekday* = Friday and *TripType* = 30).

Your goal is to predict the TripType for each customer visit. As will be described in Section 3 below, the customer visits contained in the `train.csv` file will be split into two sets: (i) the training set for which the *TripType* values will be known and (ii) the test set for which *TripType* values will not be known and you will have to predict them. You should choose a data representation that takes into account all the purchases and returns of a single customer visit. That is, each row of your data matrix should represent a single trip (e.g. the two lines of visit 7 should be merged into one line corresponding to that visit).

Note that some values inside the `train.csv` are missing and you should somehow handle them. You can simply ignore those products. Alternatively, you can create a new value that indicates that a value is missing. Otherwise, you can replace them with the most common values of that field.

In addition, some features offer no valuable information for classifying visits into trip types. For example, the *VisitNumber* field is simply a counter of the visits and can therefore be ignored. In this way, we can eliminate irrelevant features and maintain only those that are important for the classification task.

After reading the file, the emerging features will be of string type. In order to apply some classification algorithms, it is necessary to replace these strings with integers or floats. For example, you can replace the days of the *Weekday* field with integer values (e.g. 0 for Monday, 1 for Tuesday, etc.) so that your algorithms can process them. Another way of representing categorical variables is the so-called one-hot encoding which uses one feature for every possible state of the variable and depending on its current value, one of those features is set to 1, while all the others remain 0. For example, in the case of the *Weekday* attribute, 7 new features will be created to represent the attribute (e.g. 1000000 for Monday, 0100000 for Tuesday, etc.). Note that in the one-hot encoding representation, the number of produced features is equal to the number of the different values taken by the initial feature.

You can apply feature selection techniques in order to reduce the number of features and construct a reliable classifier. You can also create new features that may help in classification. You can also perform dimensionality reduction and investigate whether its application improves the classification results. Moreover, you can combine noisy features to produce new features that provide more valuable information.

To classify visits into trip types, you can write your own classifier. However, you are recommended to use the classifiers and preprocessing methods provided by the `scikit-learn`[3] library. `scikit-learn` is a very powerful machine learning library in Python which can be used in the preprocessing step (e.g. for feature selection) and in the classification task (a plethora of classification algorithms have been implemented). For example, given a training set $trainSet$ and its labels $trainLabels$, and a test set $testSet$, `scikit-learn` allows us to predict the labels of the instances of the test set using a Decision Tree classifier with the following code:

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf.fit(trainSet, trainLabels)
predictions = clf.predict_proba(testSet)
```

We make use of the function $predict\_proba()$ that returns the probability of each instance belonging to each class because as will be discussed below, the evaluation will be based on the logarithmic loss.

3 Evaluation

To evaluate the models you will build, we will use cross-validation[4]. Cross-validation is a method for assessing how the results of a statistical analysis will generalize to an independent dataset. It is mainly used in classification problems where someone wants to know how a model will behave in practice. Suppose you have a model with one or more parameters and a dataset on which you can train the model. The training process optimizes the model parameters so that the model classifies the training data as accurately as possible. If you then use the model to make predictions on an independent set, you will notice that the model does not achieve as high accuracy as on the training set. This is called overfitting and is very likely to occur when the training set is small or when the model consists of several parameters. Cross-validation provides a way to measure how the model will generalize to an independent dataset.

To evaluate the performance of your classifiers, you will use k-fold cross validation with $k = 10$. According to this method, the original dataset is randomly split into k subsets of equal size. One of the k subsets plays the role of the test set, while the remaining $k - 1$ subsets act as the training set. The process is repeated k times, and each of the k subsets is used exactly once as the test set and the remaining $k - 1$ as part of the training set. The k results can then be combined together to produce an overall evaluation of the model.

For the evaluation, you will use the multi-class logarithmic loss. For each visit, you must predict 38 probabilities (one for every *TripType*). Given these possibilities, the logarithmic loss is calculated as follows:

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij})$$

where N is the number of visits in the test set, M is the number of trip types, $log()$ is the natural logarithm, y_{ij} is 1 if observation i is of class j and 0 otherwise, and p_ij is the predicted probability that observation i belongs to class j. An ideal (perfect) model would return logarithmic loss equal to 0. In practice, it is very hard to achieve performance comparable to that of an ideal model, and hence, the built models return higher logarithmic losses.

If, for your own interest purposes, you want to participate in the Walmart competition, you can register on the Kaggle platform and download the test set which is available on the website of the

[3] `http://scikit-learn.org/`

[4] `http://en.wikipedia.org/wiki/Cross-validation_(statistics)`

competition[5]. You can then use your model to predict to what trip type each visit belongs, submit the results file on the platform and you will be given the logarithmic loss you achieved and your rank compared to the other participants.

4 Provided Source Code

You are given an initial Python code that uses only information about the day on which each visit took place in order to predict the type of the trip. Using a losistic regression classifier, the returned logarithmic loss under 10-fold cross-validation is equal to 3.074. As part of this assignment, you are asked to modify the code in `main.py` and `classify.py` to predict the trip types that customers took to get to the store.

4.1 Description of `main.py`

Below, we give a description of the source code in `main.py`.

```
# Load data
csv_file_object = csv.reader(open('train.csv', 'rb')) # Load in the csv file
header = csv_file_object.next()                       # Skip the fist line as it is a header
rows=[]                                               # Create a variable to hold the data

for row in csv_file_object: # Skip through each row in the csv file,
    rows.append(row[0:])    # adding each row to the data variable
data = np.array(rows)       # Then convert from a list to an array
```

The above code initially loads the data from the `train.csv` file to matrix $data$. Each row of the matrix corresponds to a purchase or return of a product. Continuous lines with same *VisitNumber* values correspond to products purchased or returned on a single visit. Keep in mind that the entries of matrix $data$ are of type string. You can convert them to integers or floats using the commands $data = data.astype(int)$ and $data = data.astype(float)$ respectively. To be able to do this, all the entries must contain only numeric characters.

```
daysIndex = {'Monday':0,'Tuesday':1,'Wednesday':2,'Thursday':3,'Friday':4,'Saturday':5,
             'Sunday':6} # A dictionary keyed by day to its index
```

We create a dictionary keyed by the days of the week to a number corresponding to their position in the X matrix that will be constructed next.

```
numVisits = len(np.unique(data[:,1])) # Number of distinct visits

X = np.zeros((numVisits, len(daysIndex))) # Matrix containing the day of each visit
Y = np.zeros(numVisits) # A matrix containing the trip types of the visits
```

Initially, we set the number of visits equal to the number of unique elements of the second column of the matrix $data$ (column corresponding to *VisitNumber*). Then, we initialize the matrix of visits (X) and the matrix containing the trip types of the visits (Y).

```
previousVisit = 0
index = -1
for i in range(data.shape[0]):
    if data[i,1] != previousVisit: # If visit number has changed, initialize a new visit
        index += 1          # The index of the new visit
        num_products = 1    # Set the number of products to 1
```

[5] https://www.kaggle.com/c/walmart-recruiting-trip-type-classification/data

```
        previousVisit = data[i,1] # Set previous visit number to the current visit
        X[index,daysIndex[data[i,2]]] = 1 # Set the index of the day of the visit to 1
        Y[index] = int(data[i,0])      # Store the type of the trip of the current visit
    else:                    # If visit number has not changed, it's still the same visit
        num_products += 1    # Increase the number of products of the current visit
```

We read one by one the rows of matrix $data$ and we store in $previousVisit$ the *VisitNumber* of the current row. We then read the next row and if its *VisitNumber* value is different from the value of $previousVisit$, we know that a new visit is under way. In a new row of matrix X, we set the element corresponding to the day of that visit equal to 1. For example, if the day of the visit is Monday, we set the first element of the row to 1 and all the other elements remain 0. Similarly, we set the next element of vector Y equal to the trip type taken by the customer.

```
kf = cross_validation.KFold(X.shape[0], n_folds=10) # Initialize cross validation
```

We use the $KFold()$ function with $n\_folds = 10$ to apply 10-fold cross-validation. For each of the 10 rounds, the function returns which rows of the matrices belong to the training set and which to the test set.

```
for trainIndex, testIndex in kf:
    trainSet = X[trainIndex]
    testSet = X[testIndex]
    trainLabels = Y[trainIndex]
    testLabels = Y[testIndex]

    predictions, trips = classify(trainSet, trainLabels, testSet)
    logloss = log_loss(testLabels, predictions, trips)
    print 'Log Loss: ', logloss
    totalLogloss += logloss
    iterations += 1
print 'Average Log Loss: ', totalLogloss/iterations
```

For each of the 10 rounds of cross-validation, we store in $trainSet$ and $trainLabels$ the training data and the categories to which they belong respectively. Similarly, we store in $testSet$ and $testLabels$ the test data and the categories to which they belong. We use the function $classify()$ (implemented in `classify.py`) to predict the probabilities of each visit of the test set to belong to each one of the 38 classes and compare them to the actual classes stored in $testLabels$ to compute the logarithmic loss for each of the 10 rounds as well as the aggregated logarithmic loss.

4.2 Description of `classify.py`

Below, we give a description of the source code in `classify.py`.

```
def classify(trainSet, trainLabels, testSet):
    clf = LogisticRegression()
    clf.fit(trainSet, trainLabels)
    predictedLabels = clf.predict_proba(testSet)

    return predictedLabels, clf.classes_
```

We first initialize a Logistic Regression classifier. We then train the classifier using the visits contained in $trainSet$ and their corresponding trip types contained in $trainLabels$. Finally, we use the $predict\_proba()$ function to predict the probabilities each visit in $testSet$ to belong to each one of the 38 trip types. Besides the predicted probabilities, the $classify()$ function also returns a vector that contains the trip types corresponding to the columns of the $predictedLabels$ matrix.

5 Details about the Submission of the Assignment

The assignment can be done in teams of 2-3 people. Your final evaluation for the project will be based on both the logarithmic loss that will be achieved, as well as on your total approach to the problem. As part of the assignment, you have to submit the following:

- A 2-3 pages report, in which you should describe the approach and the methods that you used in the project. Since this is a real classification task, we are interested to know how you dealt with each part of the problem, e.g. how you created your representation, which features did you use, if you applied dimensionality reduction techniques, which classification algorithms did you use and why, the performance of your methods (logarithmic loss and training time), approaches that finally didn't work but is interesting to present them, and in general, whatever you think that is interesting to report.
- A directory with the code of your implementation.
- Create a `.zip` file containing the code and the report and submit it to the eClass platform.
- **Deadline:** January 22, 2016.