ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

Data Challenge

Project Report By:
Gkountoumas Filippos (f3352105)
Legkas Sotirios (f3352112)
Papadopoulos Nikolaos (f3352118)

Introduction

For our Data Challenge Project, we were asked to study and apply machine learning/ data mining techniques to a real-world classification problem. The task of the challenge was to classify domain names of Greek sites and assign them into a predefined category[1]. In order to do so, we were free to choose from any available model and technique we were taught during this course. In the context of this project, we were given two different data sets. The first one was a web graph consisting of several thousands of Greek domain names, while the second was the textual content of a subset of all of those domains. Our goal was to approach this problem, similarly to any classification problem; therefore, we needed to learn the parameters of a classifier from a collection of training products (with known class information) and afterwards predict the class of the unlabeled products (test data). The actual test domains category was a set of data, that remained unknown to each team. The whole process of the project was conducted through the Kaggle platform in the form a Kaggle Challenge[2].

Quick overview of the implementation process

1. Gathered and imported all the data
2. Created a data frame from the training data with 3 features (site name, textual information, category)
3. Created a bigger data frame containing all the domains that have textual information
4. Processed and cleaned the textual information
5. Crated usable vectors using TF-IDF
6. Reduced the dimensionality using Truncated SVD
7. Created the Graph using the StellarGraph library
8. Split the train data to train, development sets
9. Fit the training data to the model and evaluate using the evaluation set
10. Predict for the test and fine tune according to the results
11. Error Analysis

[1] After processing the given data using data mining techniques, the categories were (to the best of our knowledge): 0) E-shop, 1) Sports, 2) Entertainment, 3) Magazine-News, 4) Education, 5) Government, 6) Company-sites, 7) Travel, 8) (couldn't figure out), 9) Medical

[2] We need to note as this point that the evaluation of the model performance will be assessed using the multi-class logarithmic loss measure. This metric is defined as the negative log-likelihood of the true class labels given a probabilistic classifier's predictions.

Preparing the Data

The preprocessing of our data started by reading the training data and visualizing the balance of the classes. As we can see from figure 1 the training data are imbalanced.

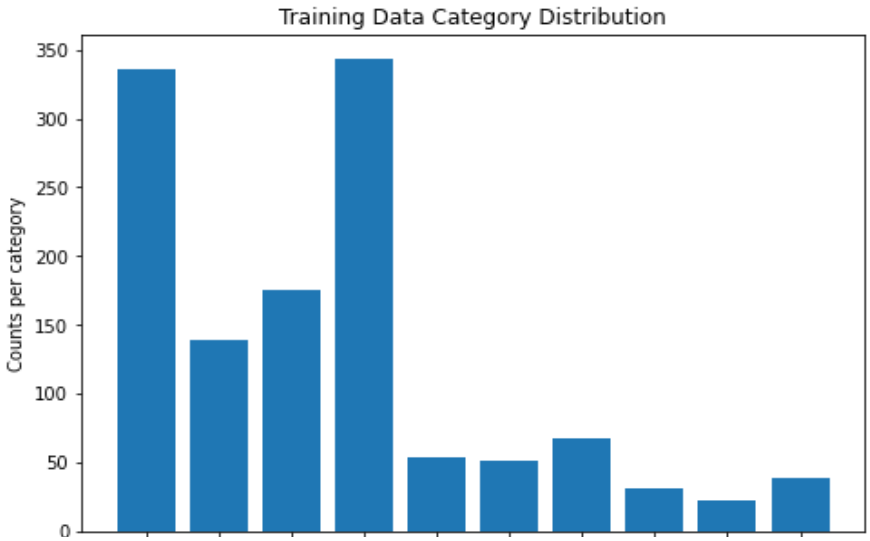


Figure 1 Training Data Class Balance

Data Cleaning

Subsequently we had to make a choice. We would either use the textual information of just the training data, or we could use every available text given to us. We began with the first, but after evaluating the model, we decided to use the second as it yielded better results. Afterwards it was time to clean the textual information to reduce its size but also cut out the non-influential parts. We started by creating a function (named prep) to process the text and do a set of changes such as: 1) Remove the HTML tags, 2) Remove punctuation and symbols, 3) Remove numbers, 4) Lowercase, 5) Strip the accents, 6) Remove a Greek Stopwords, 7) Remove words under 4 letters, 8) Remove multiple spaces. Using the processed textual information, we achieved better scores but since there many quite similar words we finally decided to also use a Greek Stemmer function which helped our models achieve the best scores. After researching the available texts, we saw that lengths of the textual information had a significant difference in variance. Therefore, we decided to keep only the 2500 words of each text, which was very close to the average text length.

Feature Engineering

In this part of the process, we needed to formulate the texts in a way that would allow us to fit out data in our models. We used two different techniques, which surprisingly, yielded almost similar results, but in our code, we kept the one which gave us the lowest score. The two aforementioned techniques were:

1. **Term frequency – inverse document frequency** (TF-IDF)
   We used TF-IDF in order to transform our text into usable vectors. For the set vectors we allowed only words with a minimum frequency of 50 and a maximum frequency of 300. After our vocabulary was formed, we only kept the 2500 of them as max features. For this technique we processed the textual data, one step further by using a Greek Stemmer.

2. **Greek Word Embeddings**
   In order to transform our data we used the Greek word embeddings of AUEB. We used the processed textual data, but without stemming them, or removing the accents.

In both cases after the creation of the features and some experimentation with the models, we decided to reduce the dimensionality of the features and make more compact ones (in terms of valuable

information), so we used **Truncated SVD** to transform the features into 128 dimensions. The number of dimensionality reduction was constantly changed, until the best results were achieved using 128.

In order to use the given data to their maximum capacity, we decided to also create a Graph using the edgelist file provided to us (the file contained all the Greek domains as nodes and their respective relationships). In order to formulate the features, we used Node2Vec. This way we could help our model, gain insights of domains, with not much textual information, or categories with a very small number of entries (such as categories "7","8","9").

3. **Node2Vec**
   Node2Vec is an algorithmic framework for representational learning on graphs. Given any graph, it can learn continuous feature representations for the nodes, which can then be used for various downstream machine learning tasks. For our task we used a BiasedRandomWalk, with a maximum random length of a walk set to 15, also we set the maximum number of random walks per root node to 15. Node2Vec also uses probabilities of returning to source code (p) and moving away from source node (q). We set those parameters in a way we thought it would lead to our walks staying "in the neighborhood", as this way we could find relations of domains of the same neighboorhoods.

Mode Building

After the first valuable steps of the process, it was time to start building models that could use our processes data to return the desired results. In this part we had to choose which implementations we could use so we decided to try many of the implementations that were taught during the Course's labs.

Logistic Regression

First of we started with the baseline models, using only the textual information, or the Graph information of the domains. It was apparent that such a simple model like this, even after using hyperparameter tuning using GridSearch (for solver, penalty, C) could not result in a desired score. (The best model using only Logistic Regression scored 1.09 during the submission of the evaluations). Afterwards we decided to use the Graph baseline using only the nodes information of the graph which was even worse than the previous one. Finally, we combined the two baselines by providing both textual and node information which resulted in a score of 1.01 during the submission, which was a far better results that any of the baseline models on each own. That result confirmed our suspicion that a combined use of all the available information in any model would lead in optimal scores.

Graph Neural Network (GNN)

After the baseline models, we decided to use a GNN, which is a class of neural networks for processing data sets represented by graph data structures. In order to test this model, we used the provided code from the course trying to tune some of it's hyperparameters (number of hidden layers, units of hidden layer, learning rate). After the tuning the GNN yielded much better results than the Logistic Regression with the best submission scoring 0.90 on the submission to Kaggle. After many trials, and using our error analysis we concluded that we needed to use a different architecture to achieve even better results.

Graph Convolution Network (GCN)

Finally, after trying different implementations of models and not getting any better we finally decided to check the literature for any other architecture that could lead to a better model. Upon searching we found the StellarGraph library that supports many state-of-the-art machine learning algorithms on graphs. A machine learning model in StellarGraph consists of a pair of items. First are the layers themselves (graph convolution layer, dropout layers, conventional dense layers) and second is a data generator to convert the core graph structure and node features into a format that can be fed into the Keras model for training or prediction. We implemented this model using 2 convolution graph layers, followed by 2 dropout layers and

finally one dense layer with 10 units (because we have 10 categories). After trial and error, we tuned the model to a set of specific hyperparameters that gave us the best result, with a submission score of 0,72 and consecutive scores under 0.78. The exact structure of our model is as follows[3]:

- 2 Convolution Graph Layers (32 units on each layer, activation function: ReLU)
- 2 Dropout Layers (with dropout rate of 0.5)
- 1 Dense layer (10 units, activation: SoftMax)
- Loss (categorical cross entropy) and optimizer (Adam with a learning rate of 0.1)

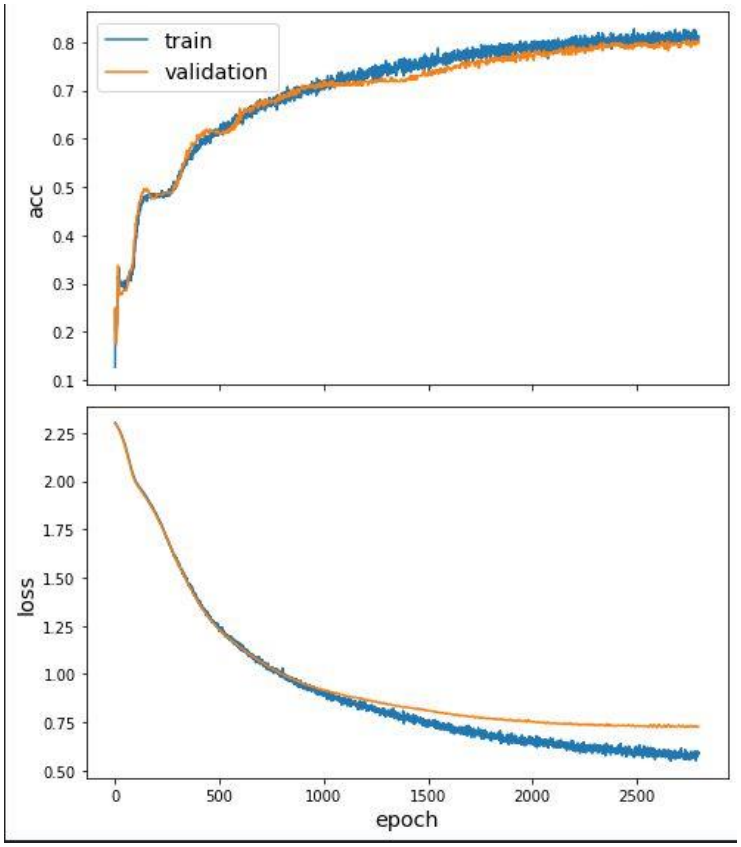
Model Evaluation & Error Analysis

After training our model with the best set of hyperparameters, it was imperative to show the results and analyze them in order to find possible faults.

Learning Curves

Learning curves are a widely used diagnostic tool in machine learning for algorithms that learn from a training dataset incrementally. It is basically a line plot of learning (y-axis) over experience (x-axis). The model can be evaluated on the training dataset and on a holdout validation dataset after each update during training and plots of the measured performance can created to show learning curves. Reviewing learning curves of models during training can be used to diagnose problems with learning, such as an underfit or overfit model, as well as whether the training and validation datasets are suitably representative. For our project we used two plots, one Optimization Learning Curve based on the metric by which the parameters of our model are being optimized (loss).

Figure 2 Learning Curves of the final model (GCN)



After plotting the learning curves, we started diagnosing our model. The shape and the dynamics of the learning curves in our occasion suggest that there are no signs of underfitting[4] since the training and validation loss curves are close to each other, therefore the complexity of our model was enough to learn the training data. There are also no signs of excessive overfitting[5], since the learnings curve of validation doesn't show an increasing trend even after the separation from the training curve. The constant fluctuations although, could suggest that our train dataset shows signs of an unrepresentative dataset, meaning that it does not provide sufficient information to learn the problem, relative to the validation dataset used to evaluate it. This situation could be identified by a learning curve for training loss that shows improvement and similarly a learning curve for validation loss that also shows improvement, but a gap remains

[3] Note: For this implementation we used the TF-IDF instead of the Greek Word2Vec reducing the initial text to 2500 words and using Truncated SVD reducing the feature dimension to 128.
[4] Underfitting refers to a model that cannot learn the training dataset.
[5] Overfitting refers to a model that has learned the training dataset too well, including the statistical noise or random fluctuations in the dataset

between both curves. In our case due the nature of the dataset and of specific categories this could be the exact case.

Confusion Matrix

To further investigate and evaluate our models results, we decided to create 2 confusion matrices, one with the absolute values of the data and one with the respective percentages as shown in the figures 3,4 below.

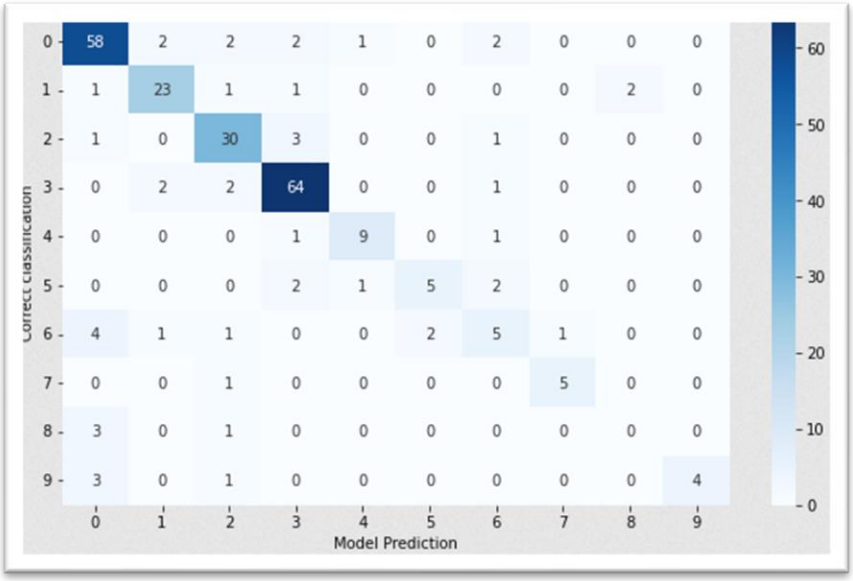


Figure 3 Confusion Matrix with absolute values

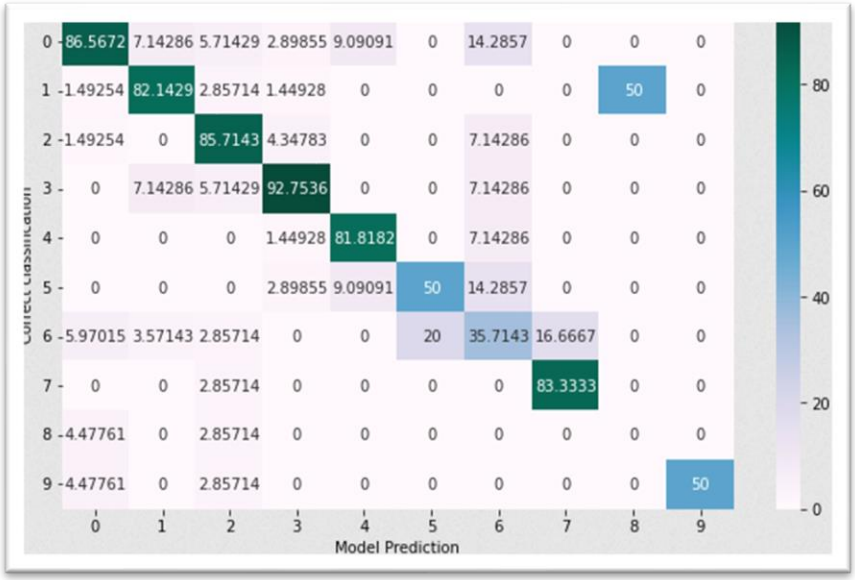


Figure 4 Confusion Matrix with percentages

As we can see from the confusion matrices, it was apparent that the categories with the most data, were the ones classified with the highest accuracy (such as categories 1,2,3), whilst the categories with the lowest amount of available data, were more difficult to be classified on the correct category. Especially category '8' of our data was the worst one in terms of classifying domains to it, which at first sight made perfect sense since we had very little data about the specified category. In order to validate our findings, we decided to check manually some of the sites belonging to the set category. What we came to understand was that domains belonging to that category (such as 'iphonehellas.gr','physics4u.gr','ivf-embryo.gr'), were domains that did not have any logical correlation between them. As a matter of fact, they were so distant that even a human annotator would not be able to classify a site belonging to the aforementioned class.

Conclusion

Throughout the project we were able to learn and experiment on datasets that are not one dimensional and combine different representations of the same information (such as text and graphs), and were able to combine those two to improve our model's overall performance. One more important fact that we came to learn throughout this process, is the fact that many times achieving the best result depends on many different aspects of the implementation (such as the quality of the dataset and the available resources). Overall it was a great and very informative experience and the fact that this was our first ever Kaggle Challenge(and definitely not our last), motivated us to keep pushing over our limits in order to learn new and more advanced techniques that could advance our knowledge and give us a competitive advantage.