|  |  |
| --- | --- |
|  | **DOKUZ EYLÜL UNIVERSITY**  **ENGINEERING FACULTY**  **DEPT. OF COMPUTER ENGINEERING** |

# CME 4416 Introduction to Data Mining

# Term Project Report

**2019-2020 SPRING**

**2016510086**

**Mustafa ÖZSARAÇ**

# 1-Introduction

In this progress report, I will explain my process for the term project. This report will contain explanation of what I have done, description of my data set and what techniques I applied in order to prepare the dataset. Also, the algorithm that I choose to apply for this dataset and problem. I have chosen “**document classification**” as my topic for this project. My aim for this term project will be to create a system which can accurately classify these BBC documents into the right categories.

# 2-Information About Dataset

Text documents are one of the richest sources of data for business. In this project I will use a public dataset from the **BBC** comprised of 1490 articles, each labeled under one of 5 categories: **business, entertainment, politics, sport or tech**. This is a great dataset for document classification, also for classification algorithms which I will use for this project. The dataset has 1490 rows in it. I’ll use two test techniques, which are K-Fold Validation and train-test split (%90-10). The goal will be to build a system that can accurately classify previously unseen news articles into the true category.

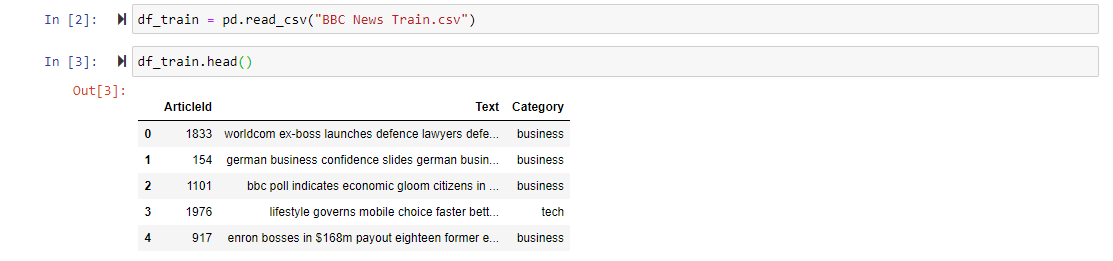
# 3-Data Preparation Techniques

I used **Python** with **Jupyter Notebook** and “**pandas**” library to load this data as data frame and add a category id (which I will do later on). The **Dataframe** is a useful data structure, first popularized by the R language, that allows us to easily transform and navigate our dataset in an efficient manner.

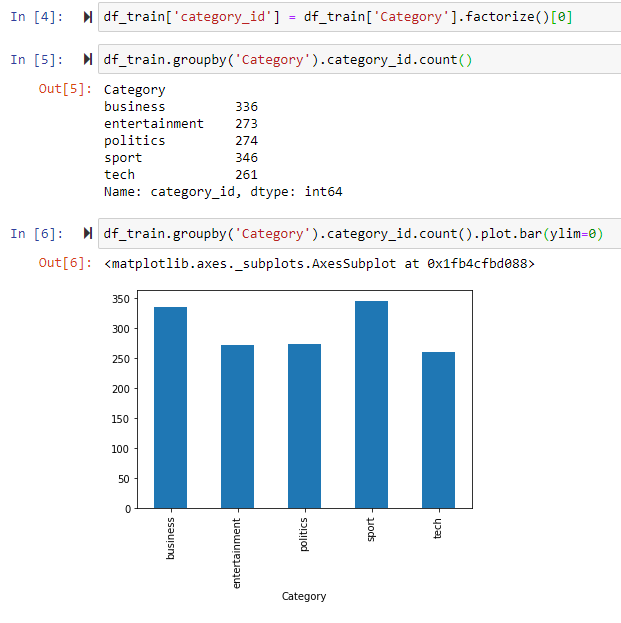
Firstly, I will add necessary libraries such as **pandas**, **numpy** and **matplot** as you see below:



Then, I used pd.read\_csv to read the csv files to a dataframe and printed its head(firs 5 rows) as shown below:



After that, before diving into the classification algorithms I thought it would be better if I get familiar with structure and characteristic of my dataset. First, it’s always good to see the number of documents per class. In order to do that, I grouped the categories and printed the numbers also the histograph of this numbers. This is how I did:



Here, we see that the numbers of articles per class is **roughly balanced**, which is good for me. If my dataset were imbalanced, I would need to carefully configure my model such as **undersampling or oversampling** for each class.

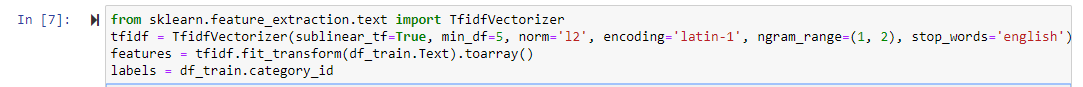
To further my analysis, I need to transform each document’s text to **a feature vector**, which is a list of numerical values representing some of the text’s characteristics. This is mostly because most of the machine learning/data mining algorithms work better with **numerical values rather than the raw texts.**

One common approach for extracting features from text is to use the **bag of words** model: a model where for each document, an article in my case, the presence the frequency of words is taken into consideration, but the order in which they occur is ignored. Specifically, I will calculate a measure called **TF-IDF** **(Term Frequency, Inverse Document Frequency)** for each term in my dataset. These statistics will show the words’ importance for each document. I use the words’ frequency as proxy for importance.

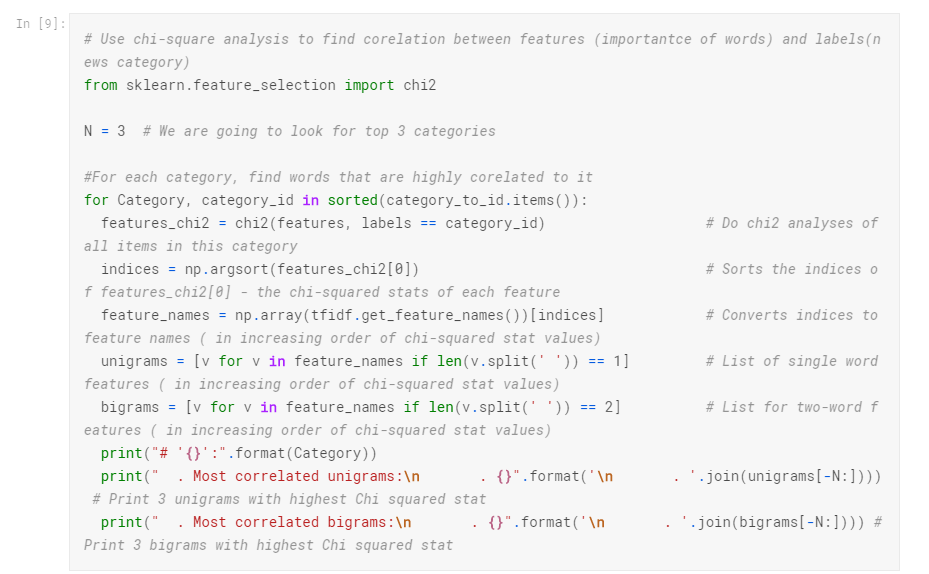
For example: If word “**basketball**” is mentioned 30 times in a document, it may be more important than some other word that was mentioned only once.

I’ll also use the document frequency (the number of documents containing a given word) as a measure of how common the word is. This minimizes the effect of **stop-words** such as pronouns, or domain-specific language that does not add much information (for example, a word such as "news" that might be present in most documents).

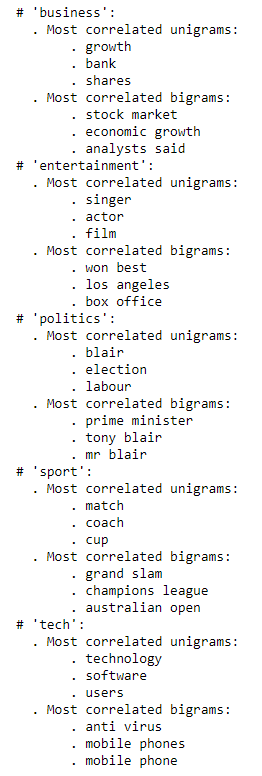
I’ll use **sklearn** a machine learning library which is accessible for everyone. Within sklearn I will use **TfidVectorizer** class to calculate **tf-idf** values for each of document in dataset. Here, is a code in python of how I did this:



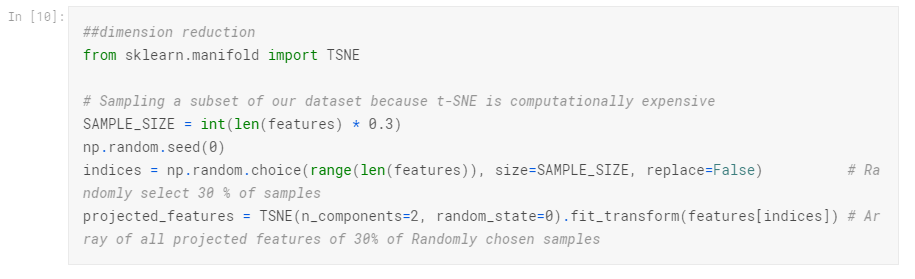
As you see, I passed some additional parameters to the **tfidf** class: **sublinear\_tf** means that I will use the log of the frequency, as the words’ frequencies follow an Expo distribution and normalize my vectors **to l2 form** so that the length of the document does not bias its representation. Also by doing this I consider **bigrams** (2 words) as these might carry a different meaning than each other of their components separately (e.g box office vs box and office).

The resulting **“features”** variable contains one row of numerical features (each representing the **tf-idf** for a word or pair of words) for each of our documents. This representation will be useful for solving classification task, but also we can use **Chi-Squared Test** to find the terms are most correlated with each of the categories.

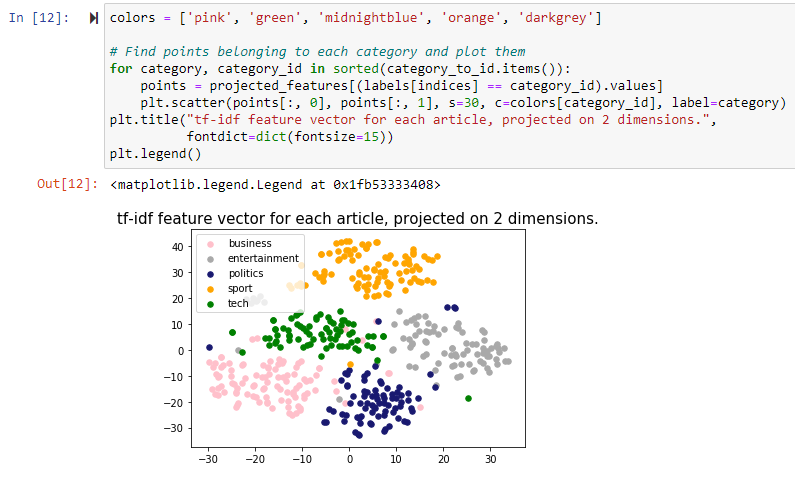
And the output is as shown below:



After that, we can also use **dimensionality reduction techniques**, such **t-SNE** to project our high dimensional tf-idf features into a **2D plane**, where they can be visualized. This is done by keeping **nearby points** in the high-dimensional space close to each other in the projected space.



After finding these projected features, to visualize them in a 2d plane, usage of the code as shown below would be correct:



Now, you can see that different categories fell in different areas, which means that we can expect high accuracy performance of classification.

So, I’ve done **data extraction and data preparation steps**. Now I’m more familiar with my data set and I can move on to select proper classification algorithm to apply.

# 4-Choice of Metric and Algorithm – Model

Firstly, I need to choose which **metric** to optimize for my dataset and problem. Here, I am dealing with a **multiple class classification** task. Given the relative balance of dataset, **accuracy** would be a proper metric for our dataset and classification problem. If one label was more important than the others, we should use **precision and recall** for each class.

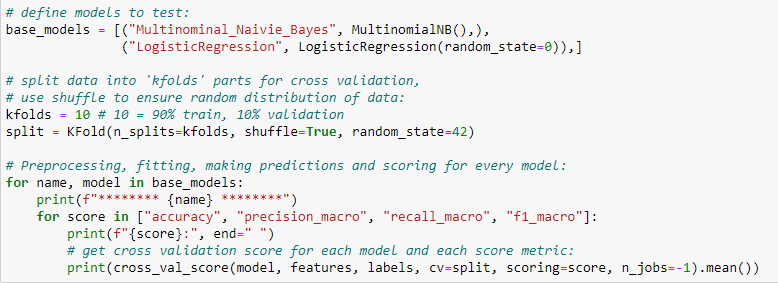
For the choice of model, I choose **Multinomial** **Logistic Regression**. It’s mainly because when I looked up for multi-class classification problems on the internet, majority of experts recommend **logistic regression** and **Multinomial NB** due to its high median accuracy dealing with this (multi-class classification) problem.

Now, I will train and test these models and show their results.

# 5-a Model Tests and Results(Python)

**With KFold:**

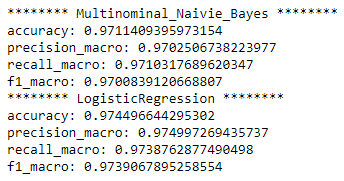
At first, I wrote the code shown below:



This code does as follows:

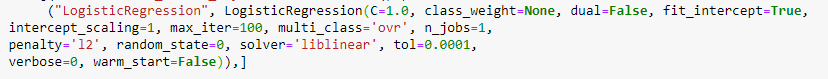
* In here I defined two models, which are MultinomialNB and MultinomialLR.
* Then I selected, kfolds which is 10, and created a split with KFold function.
* After that, with the for loop I travel all the models, and print their results with the appropriate scoring metrics. Scoring metrics consist of **accuracy, precision, recall and f1 score.**

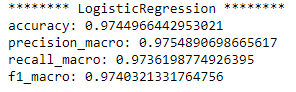
And result of these two is shown below.



As you can see from the result, **multinomial logistic regression** is slightly better.

**Parameter Tuning for Logistic Regression with KFold:**





As expected, there’s only a slightly increase in **precision and f1** but not much.

**With Train - Test Split:**

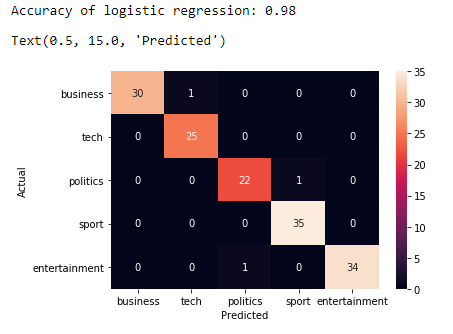
Now, I’ll test what these models would do with train test split.



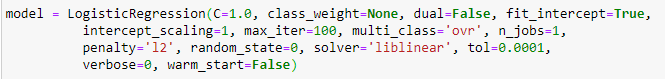
With this code,

* I created logistic regression model,
* Split the data into train and test with the ratio of %90-10.
* Train and predict, print the results.
* Print the confusion matrix.

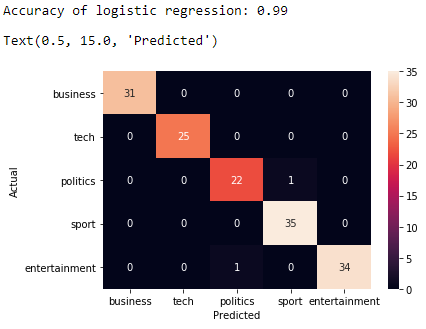
And result is as shown below:



**Parameter Tuning for Logistic Regression with Train – Test Split:**

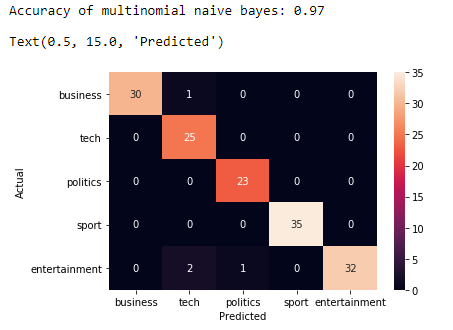


And results are as shown below:



**So yes! There’s an increase in accuracy. Now, it’s %99. Almost perfect.**

Also, I tried the train test split with Multinomial NB too. Result is as shown below:

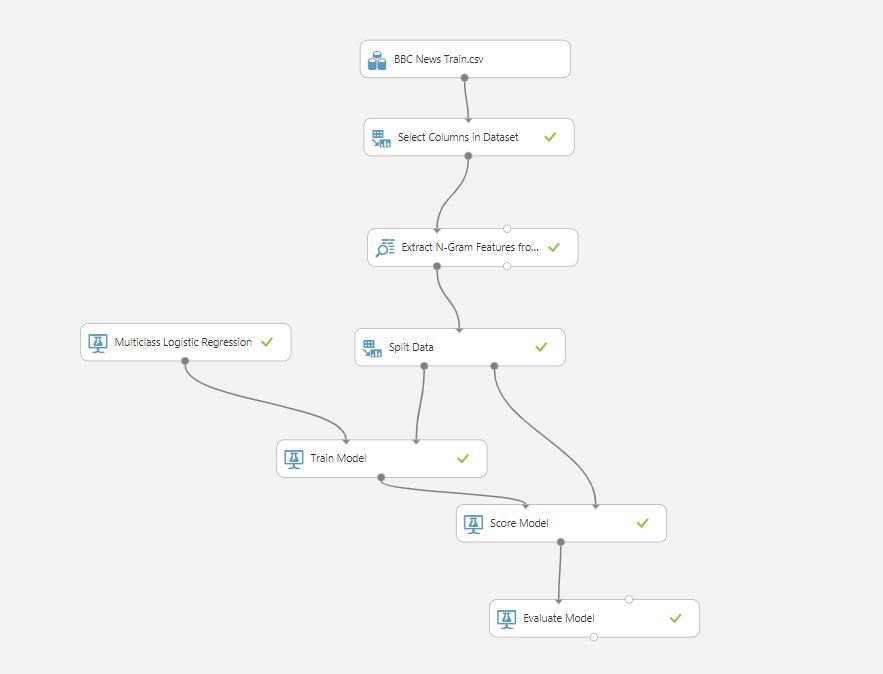


As you can see, **logistic regression is better with the train test split too.**

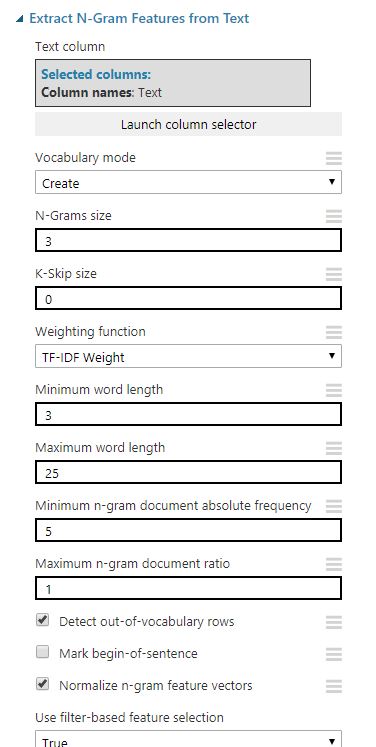
# 5-b Azure Machine Learning Studio Test and Results

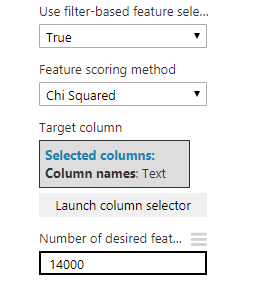
**Train – Test Split:**

This my algorithm for Azure ML Studio:

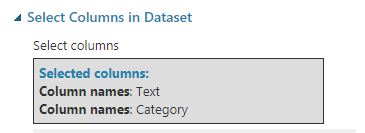


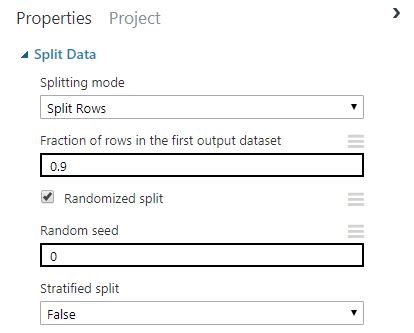
* At first, I imported my dataset which is BBC News.
* Then I selected columns which are text and category.
* I used N-Gram extraction with **TF-IDF weighting and Chi-Squared feature scoring** method.
* Then, I split the data into **train and test with the ratio of 90-10.**
* After that, I trained my Multiclass LR model and score it and print these values.

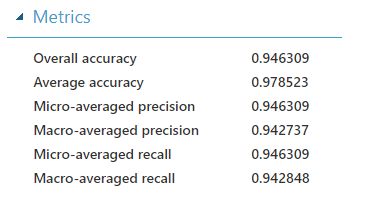
My N-Gram Parameters:

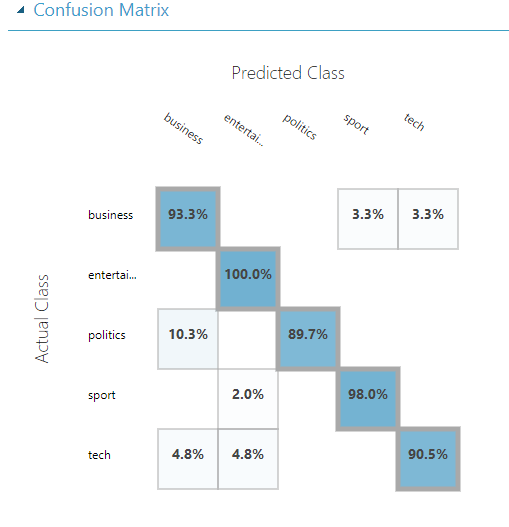


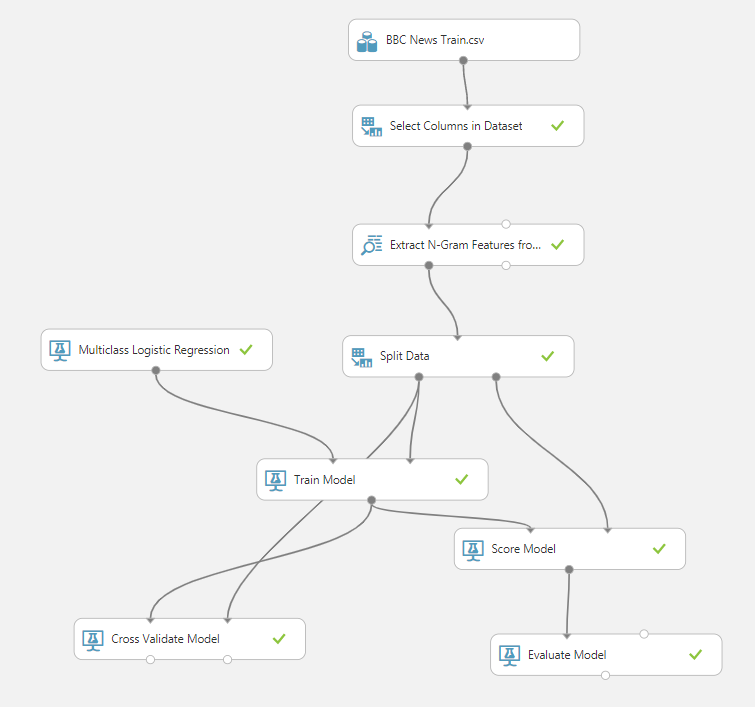
Select Columns:



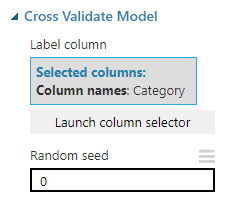
Split Data:

**Confusion Matrix and Results:**



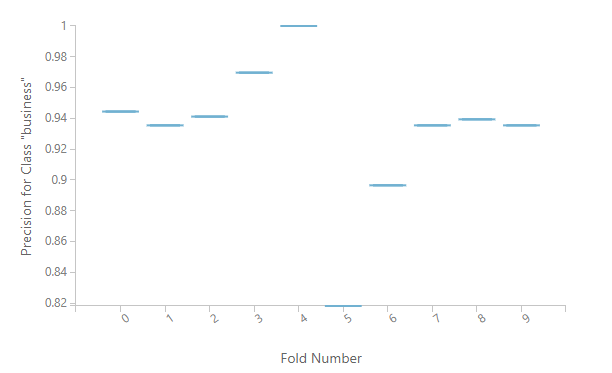
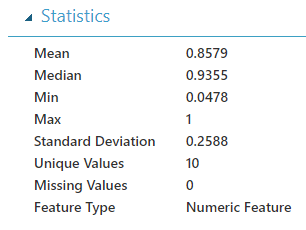
**Now Azure with cross-fold validation:**

Cross validation parameters:

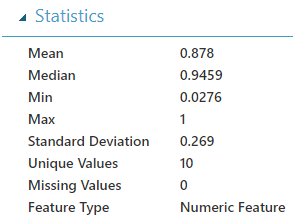
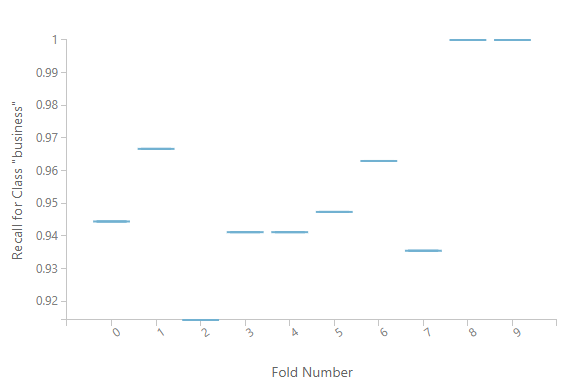


**Cross Validation Model Evaluation Results by Fold**

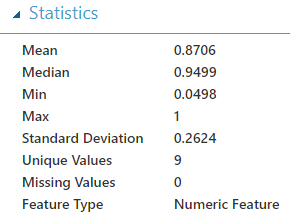
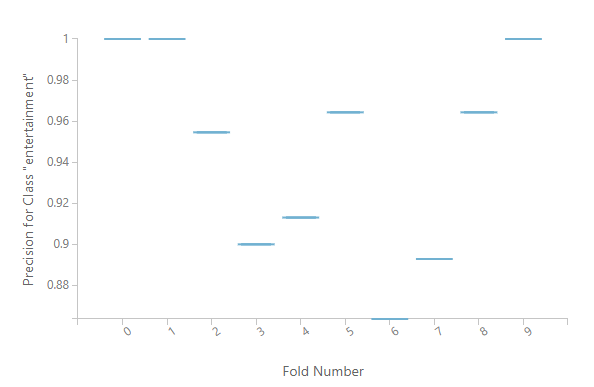
**Precision for Class Business:**



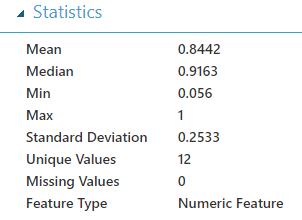
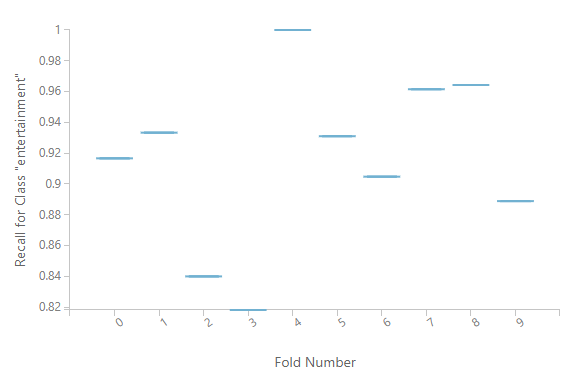
**Recall for Class Business:**



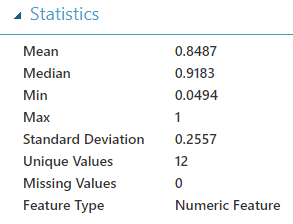
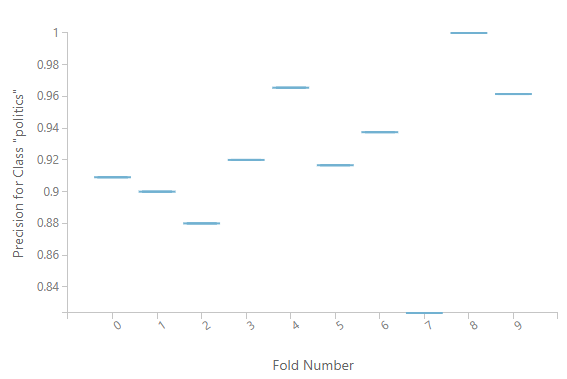
**Precision for Class Entertainment**:



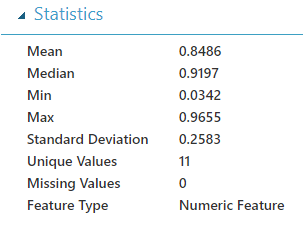
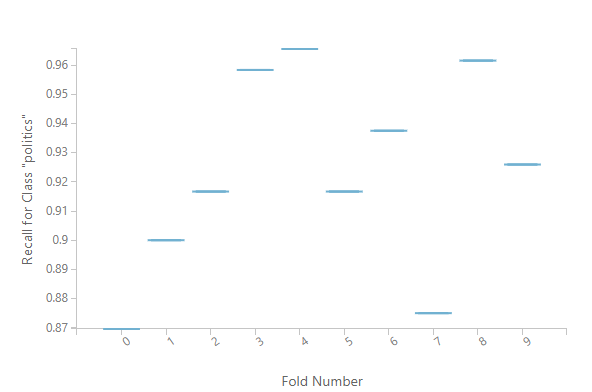
**Recall for Class Entertainment:**



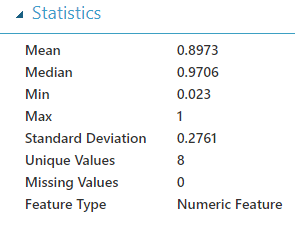
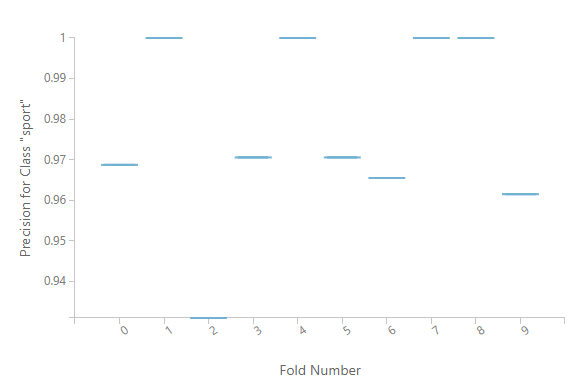
**Precision for Class Politics:**



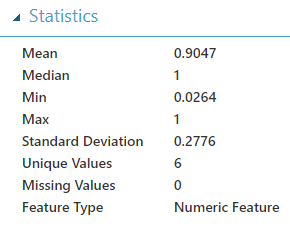
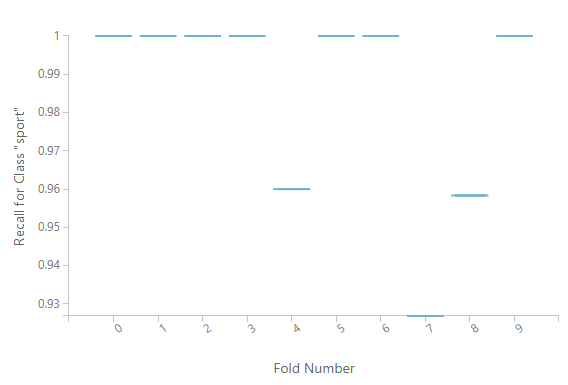
**Recall for Class Politics:**



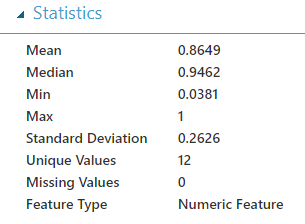
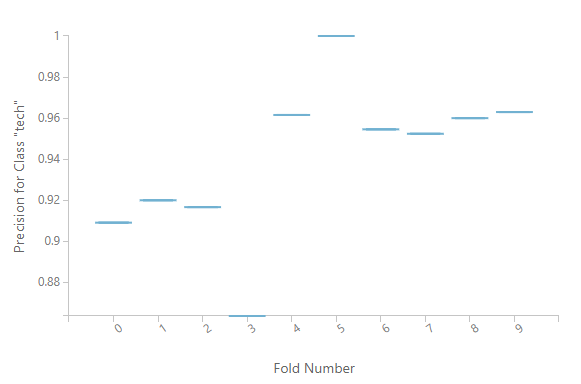
**Precision for Class Sport:**



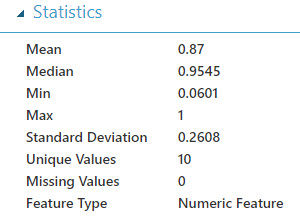
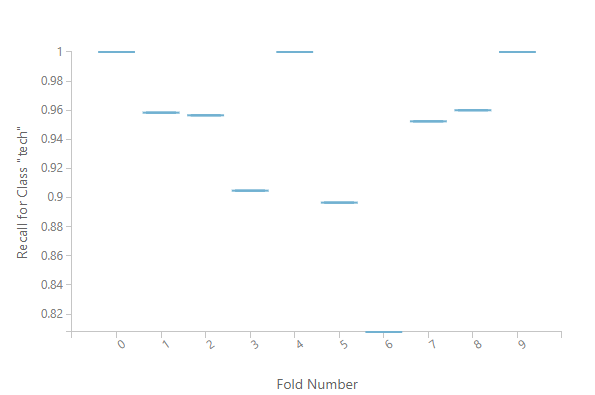
**Recall for Class Sport:**



**Precision for Class Tech:**



**Recall for Class Tech:**



# 6-Conclusion

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Recall(macro) | Precision(macro) | F1-score(macro) |
| Multiclass Logistic Regression | 0.9744 -0.99(par. tuning) | 0.9736 | 0.9754 | 0.9740 |
| Multiclass Naïve Bayes | 0.9711-0.98(train-test) | 0.9710 | 0.9702 | 0.9700 |
| Multiclass Logistic Regression  (Azure) | 0.9463 | 0.9428 | 0.9427 | 0.9427 |

As you can see from the conclusion table, **Logistic Regression with Python and train – test split** performs the best.

Its accuracy increases up to 0.99 when we’re using train test split and appropriate parameters.

Thus, Logistic Regression model is the best for this data set. I also think that Python (with Anaconda) performs better than Azure Machine Learning Studio.

Also, I think train – test split works better than cross fold validation in every try I did. Both in Python and Azure Machine Learning Studio, train – test split performed better than cross fold.

# References

1. <https://www.kaggle.com/c/learn-ai-bbc> – Dataset Source
2. <https://www.kaggle.com/abbahaddou/bbc-automatic-document-classification>
3. <https://cloud.google.com/blog/products/gcp/problem-solving-with-ml-automatic-document-classification> - Google Cloud: Problem-solving with ML: automatic document classification