capstone-proposal

October 21, 2018

1 Machine Learning Engineer Nanodegree

1.1 Capstone Proposal

Felipe Santos
October 4th, 2018

1.2 Proposal

My proposal to the capstone project is beating the benchmark in the Tradeshift Text Classification on Kaggle Competition, in this competition, the machine learning engineer has to classify text blocks in documents to certain labels, being a multiclass classification problem with tabular data. This competition started on 10/02/2014 and ended on 11/10/2014 and today is an featured competition.

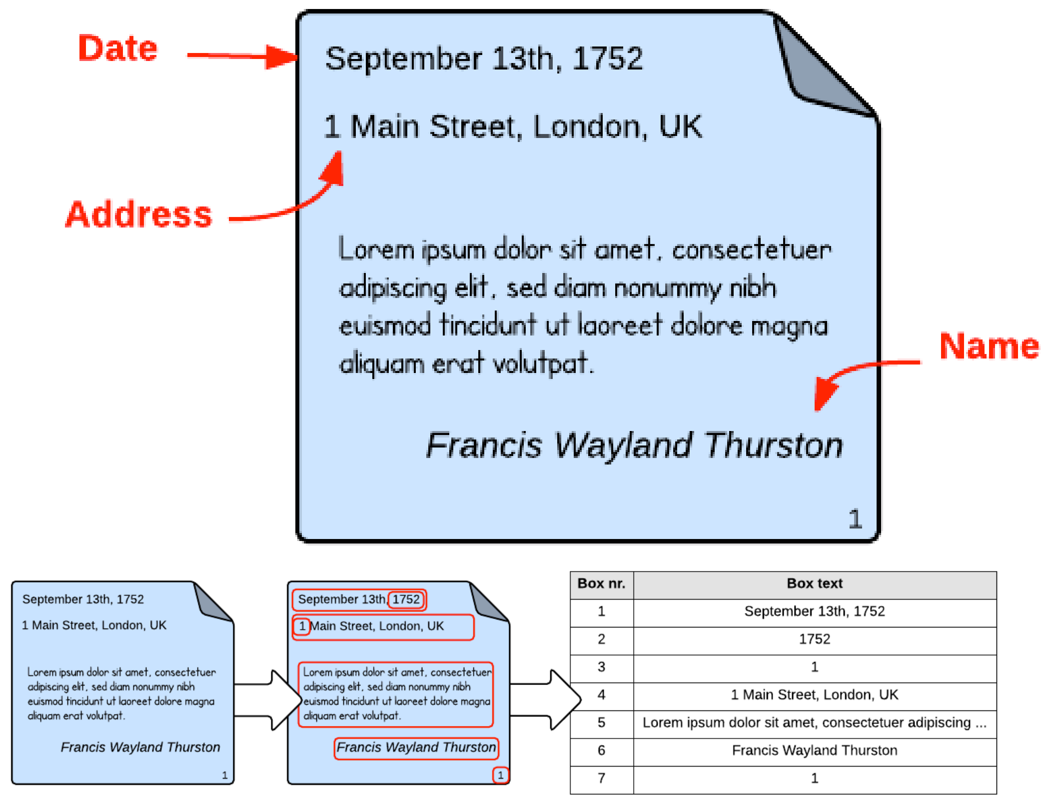
1.2.1 Domain Background

"Optical character recognition (also optical character reader, OCR) is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo" [1]. This method is used as an entry method so the document became more easily stored, compact and searched. But the process only does some dummy translate from image to text so text classification algorithms come to give us information about this unstructured format and transform from document retrieval to knowledge discovery [2]. The need of automatically retrieval of useful knowledge from the huge amount of textual data in order to assist the human analysis is fully apparent [3].

Tradeshift competition is about predicted the probability that a piece of text belongs to a given class. The dataset was created from thousands of documents, representing millions of words. In each document, several bounding boxes with text inside are selected and features are extracted from this texts and labels are assigned. For the text extraction process is used OCR (optical character recognition) and the supervised machine learning method is used to gain information and classify the text, the dataset is previously performed the OCR text extraction process and the features are already extracted. I want to learn about this project to gains insights into a future project of my own that have some similarities with this competition.

1.2.2 Problem Statement

In this competition, we have to create a supervised machine learning algorithm to predict labels from the text that is parsed from OCR and the features give to us from Tradeshift dataset. For all the documents, words are detected and combined to form text blocks that may overlap to each other. Each text block is enclosed within a spatial box, which is depicted by a red line in the sketch below. The text blocks from all documents are aggregated in a data set where each text block corresponds to one sample (row).



1.2.3 Datasets and Inputs

The files with the dataset used for this capstone is on the link in the section Data. We have 4 files on the link, all in the csv format with a 1-row header and each row stores a different sample and each collumn is separeted with comma: - **train.csv**, contains all features for the training set; - **trainLabels.csv**, contains one row per label per sample and the order of the rows is the align with the train.csv; - **test.csv**, contains all features for the testing set; - **sampleSubmission.csv**, contains a sample submission to the kaggle competition.

This dataset has ~2.1M samples with 80% as training set and 20% as the testing set, compounding of 145 features and having 33 labels to classify. The test set is split into public (30%) and private (70%) sets, which are used for the public and the private leaderboard on the competition. The features of the dataset goes to one of these categories: - **Content**: The cryptographic hash of the raw text. - **Parsing**: Indicates if the text parses as number, text, alphanumeric, etc. - **Spatial**: Indicates the box position, size, etc. - **Relational**: Includes information about the surrounding text blocks in the original document. If there is not such a surrounding text block, e.g. a text block in the top of the document does not have any other text block upper than itself, these features are empty (no-value).

The feature values can be: - **Numbers**. Continuous/discrete numerical values. - **Boolean**. The values include YES (true) or NO (false). - **Categorical**. Values within a finite set of possible values.

Some observations: * The order of samples and features is random. In fact, two consecutive samples in the table will most likely not belong to the same document. * Some documents are OCR'ed; hence, some noise in the data is expected. * The documents have different formats and the text belongs to several languages. * The number of pages and text blocks per document is not constant. * The meaning of the features and class is not provided.

1.2.4 Data Exploration

1. Section **??**
2. Section **??**
3. Section 1.2.4
4. Section **??**
5. Section **??**

Loading Data

In [1]:
```
%load_ext autoreload
%autoreload 2

import src.describe as d
import pandas as pd

pd.set_option('display.max_columns', None)
pd.set_option('display.expand_frame_repr', False)
pd.set_option('max_colwidth', -1)

train_features = d.read_train_features()
```

First look

In [2]:
```
train_features.shape
```

Out[2]:
```
(1700000, 146)
```

In [3]:
```
train_features.head()
```

Out[3]:
```
   id   x1   x2                                            x3
0  1    NO   NO   dqOiM6yBYgnVSezBRiQXs9bvOFnRqrtIoXRIElxD7g8=  GNjrXXA3SxbgD0dTRblAPO9j
1  2    NaN  NaN  NaN                                           NaN
2  3    NO   NO   ib4VpsEsqJHzDiyL0dZLQ+xQzDPrkxE+9T3mx5fv2wI=  X6dDAI/DZOWvu0Dg6gCgRoN
3  4    YES  NO   BfrqME7vdLw3suQp6YAT16W2piNUmpKhMzuDrVrFQ4w=  YGCdISifn4fLao/ASKdZFhG
4  5    NO   NO   RTjsrrR8DTlJyaIP9Q3Z8s0zseqlVQTrlSe97GCWfbk=  3yK2OPj1uYDsoMgsxsjY1Fx
```

In [4]:
```
train_features.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1700000 entries, 0 to 1699999
Columns: 146 entries, id to x145
dtypes: float64(55), int64(31), object(60)
memory usage: 1.8+ GB
```

Metadata In this section, we will categorize the collumns to try to facilitate the manipulation. We'll store: * **dtype**: int, float, str * **category**: content, numerical, boolean

```
In [5]: meta = d.create_features_meta(train_features)
        meta.head(10)
```

```
Out[5]:            role   category    dtype
        varname
        id        id      numerical   int64
        x1        input   boolean     object
        x2        input   boolean     object
        x3        input   content     object
        x4        input   content     object
        x5        input   numerical   float64
        x6        input   numerical   float64
        x7        input   numerical   float64
        x8        input   numerical   float64
        x9        input   numerical   float64
```

Extract all boolean features:

```
In [6]: meta[meta.category == 'boolean'].index
```

```
Out[6]: Index(['x1', 'x2', 'x10', 'x11', 'x12', 'x13', 'x14', 'x24', 'x25', 'x26',
               'x30', 'x31', 'x32', 'x33', 'x41', 'x42', 'x43', 'x44', 'x45', 'x55',
               'x56', 'x57', 'x62', 'x63', 'x71', 'x72', 'x73', 'x74', 'x75', 'x85',
               'x86', 'x87', 'x92', 'x93', 'x101', 'x102', 'x103', 'x104', 'x105',
               'x115', 'x116', 'x117', 'x126', 'x127', 'x128', 'x129', 'x130', 'x140',
               'x141', 'x142'],
              dtype='object', name='varname')
```

See the quantity of feature per category:

```
In [7]: pd.DataFrame({'count' : meta.groupby(['category', 'dtype'])['dtype'].size()}).reset_inc
```

```
Out[7]:     category    dtype   count
        0   boolean     object    50
        1   content     object    10
        2   numerical   int64     31
        3   numerical   float64   55
```

Descriptive Statistics In this section we will apply the *describe* method on the features splited by category and dtype to calculate the mean, standart deviation, max, min…

Numerical float variables

```
In [8]: float_features = meta[(meta.category == 'numerical') & (meta.dtype == 'float64')].inde
        float_train_features = train_features[float_features]
        float_train_features_describe = float_train_features.describe()
        float_train_features_describe
```

```
Out[8]:                  x5            x6            x7            x8            x9            x
        count  1.700000e+06  1.700000e+06  1.700000e+06  1.700000e+06  1.700000e+06  1.700000e
        mean   9.551493e-01  5.531406e-02  7.906443e-01  1.731225e-01  4.462953e-01  4.196774e
        std    5.278641e-01  1.318832e-01  3.549407e-01  3.326885e-01  3.026847e-01  2.945485e
        min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00 -1.042755e+00 -5.919283e
        25%    6.367211e-01  0.000000e+00  8.438324e-01  0.000000e+00  1.961279e-01  1.670404e
        50%    1.270115e+00  0.000000e+00  9.588627e-01  0.000000e+00  4.393339e-01  4.002242e
        75%    1.414798e+00  5.837871e-02  1.000000e+00  1.451906e-01  6.866182e-01  6.822070e
        max    2.732124e+00  9.987901e-01  1.000000e+00  1.753333e+00  1.942155e+00  7.929372e
```

```
In [9]: float_train_features_describe.loc()[['min','max']]
```

```
Out[9]:            x5       x6   x7        x8        x9       x16       x19   x20  x21       x2
        min  0.000000  0.00000  0.0  0.000000 -1.042755 -0.591928 -0.352018 -46.0  0.0 -0.5762
        max  2.732124  0.99879  1.0  1.753333  1.942155  7.929372  0.999786  14.0  1.0  7.9686
```

```
In [10]: float_train_features.isnull().any().any()
```

```
Out[10]: False
```

The features that are scaled between [0,1] are: x6, x7, x21, x37, x38, x52, x67, x68, x82, x97, x98, x112, x122, x123, x137.

So we could apply scaling on the other features depends on the classifier.

And we don't have any NaN values on this features.

Numerical int variables

```
In [11]: int_features = meta[(meta.category == 'numerical') & (meta.dtype == 'int64')].index
         int_train_features = train_features[int_features]
         int_train_features_describe = int_train_features.describe()
         int_train_features_describe
```

```
Out[11]:                  id           x15           x17           x18           x22
         count  1.700000e+06  1.700000e+06  1.700000e+06  1.700000e+06  1.700000e+06  1.700000
         mean   8.500005e+05  6.154404e+00  4.487084e+00  8.096322e+00  2.301595e+03  1.874765
         std    4.907479e+05  8.957511e+00  4.623426e+00  7.123864e+00  1.745120e+03  1.517991
         min    1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000
         25%    4.250008e+05  1.000000e+00  2.000000e+00  3.000000e+00  1.261000e+03  8.920000
         50%    8.500005e+05  3.000000e+00  4.000000e+00  7.000000e+00  1.263000e+03  8.920000
         75%    1.275000e+06  7.000000e+00  6.000000e+00  1.100000e+01  4.400000e+03  3.307000
         max    1.700000e+06  1.530000e+02  2.370000e+02  2.190000e+02  1.950000e+04  1.416700
```

```
In [12]: int_train_features_describe.loc()[['min','max']]
```

```
Out[12]:             id    x15    x17    x18      x22      x23    x27    x46    x48    x49
         min  1.0        0.0    0.0    0.0    0.0      0.0      -1.0    0.0    0.0    0.0    0.0
         max  1700000.0  153.0  237.0  219.0  19500.0  14167.0  672.0  153.0  371.0  219.0  195
```

```
In [13]: int_train_features_describe.isnull().any().any()
```

```
Out[13]: False
```

All the int numerical features are not scaled, so depending on the algorithm we have to scale the feature, we don't have any missing value. The problem here is we don't know when the feature is a categorical feature or a quantitative.

Content variables

```
In [14]: content_features = meta[(meta.category == 'content')].index
         content_train_features = train_features[content_features]
         content_train_features_describe = content_train_features.describe()
         content_train_features_describe
```

Out[14]:

	x3	
count	1451737	1451737
unique	201881	26428
top	MZZbXga8gvaCBqWpzrh2iKdOkcsz/bG/z4BVjUnqWTO=	hCXwO/JldK5zcd9ejOD1FwmEgCf96eTc
freq	51212	86750

```
In [15]: uniques = set()
         for c in content_train_features.columns:
             uniques.update(content_train_features[c].unique().tolist())
         print('total uniques words={}'.format(len(uniques)))

         # flattening all the words to count them
         all_words = pd.Series(content_train_features.values.flatten('F'))
         all_words = all_words.to_frame().reset_index()
         print('total words={}'.format(all_words.shape[0]))
         all_words = all_words.rename(columns= {0: 'words'})
         all_words = pd.DataFrame({'count' : all_words.groupby(['words'])['words'].size()}).res
         all_words.sort_values('count', ascending=False).head(10)
```

```
total uniques words=979749
total words=17000000
```

Out[15]:

	words	count
565834	YvZUuCDjLu9VvkCdBWgARWQrvm+FSXgxp0zIrMjcLBc=	392698
538278	X6dDAI/DZOWvu0Dg6gCgRoNr2vTUz/mc4SdHTNUPS38=	356811
692301	hCXwO/JldK5zcd9ejOD1FwmEgCf96eTdEVy7OtY2Y2g=	317031
376725	MZZbXga8gvaCBqWpzrh2iKdOkcsz/bG/z4BVjUnqWTO=	273502
199214	B+EJpnEbkYtLnwDQYN1dP1rcfnoCnxAjKLYwQZE07Ew=	260233
15027	+yhSY//Hpg7u0bSA7NYmcmRFgv3bF4Tw3BMHrBqaTtA=	260166
528829	WV5vAHFyqkeuyFB5KVNGFOBuwjkUGKYc8wh9QfpVzAA=	237367
264280	FExKgjj6CsbToTubdZ+kGsOmUx3gCvZVJCdZPcdPNF4=	208934
808722	oo9tGpHvTredpg9JkHgYbZAuxcwtSpQxU5mA/zUbxY8=	182455
49401	1CiKJR7D66tRwH6l6wwv0p+D/tAuoW+NdSNqPTbvDoQ=	176907

```
In [16]: content_train_features.isnull().sum()
```

```
Out[16]: x3      248263
         x4      248263
```

```
x34      50846
x35      50846
x61      32
x64      71061
x65      71061
x91      32
x94      140619
x95      140619
dtype: int64
```

On the hashed words we have 979_749 unique words on 17_000_000 (1.7kk rows x 10 collumns) words giving 5.76% of uniques words on the total words. This show us that word can have a huge impact on the classifier because we have some words multiples times. But we have to take care of the NaN values and treat them.

Boolean variables

```
In [17]: bool_vars = meta[(meta.category == 'boolean')].index
         train_features[bool_vars].describe()
         train_features[bool_vars].isnull().sum()
```

```
Out[17]: x1       248190
         x2       248190
         x10      248263
         x11      248263
         x12      248263
         x13      248263
         x14      248263
         x24      248263
         x25      248263
         x26      248263
         x30      0
         x31      0
         x32      50772
         x33      50772
         x41      50846
         x42      50846
         x43      50846
         x44      50846
         x45      50846
         x55      50846
         x56      50846
         x57      50846
         x62      70978
         x63      70978
         x71      71061
         x72      71061
         x73      71061
         x74      71061
```

```
x75      71061
x85      71061
x86      71061
x87      71061
x92      140526
x93      140526
x101     140619
x102     140619
x103     140619
x104     140619
x105     140619
x115     140619
x116     140619
x117     140619
x126     32
x127     32
x128     32
x129     32
x130     32
x140     32
x141     32
x142     32
dtype: int64
```

On the boolean values, only on 2 features we have no missing values. So we have to treat all this missing values here.

Data Quality Checks *Checking Missings Values*

```
In [18]: vars_with_missing = []
         for f in train_features.columns:
             missings = train_features[f].isnull().sum()
             if missings > 0:
                 vars_with_missing.append(f)
                 missings_perc = missings/train_features.shape[0]
                 category = meta.loc[f]['category']
                 dtype = meta.loc[f]['dtype']

                 print('Variable {} ({}, {}) has {} records ({:.2%}) with missing values'.forma

         print('In total, there are {} variables with missing values'.format(len(vars_with_mis
```

```
Variable x1 (boolean, object) has 248190 records (14.60%) with missing values
Variable x2 (boolean, object) has 248190 records (14.60%) with missing values
Variable x3 (content, object) has 248263 records (14.60%) with missing values
Variable x4 (content, object) has 248263 records (14.60%) with missing values
Variable x10 (boolean, object) has 248263 records (14.60%) with missing values
Variable x11 (boolean, object) has 248263 records (14.60%) with missing values
```

```
Variable x12 (boolean, object) has 248263 records (14.60%) with missing values
Variable x13 (boolean, object) has 248263 records (14.60%) with missing values
Variable x14 (boolean, object) has 248263 records (14.60%) with missing values
Variable x24 (boolean, object) has 248263 records (14.60%) with missing values
Variable x25 (boolean, object) has 248263 records (14.60%) with missing values
Variable x26 (boolean, object) has 248263 records (14.60%) with missing values
Variable x32 (boolean, object) has 50772 records (2.99%) with missing values
Variable x33 (boolean, object) has 50772 records (2.99%) with missing values
Variable x34 (content, object) has 50846 records (2.99%) with missing values
Variable x35 (content, object) has 50846 records (2.99%) with missing values
Variable x41 (boolean, object) has 50846 records (2.99%) with missing values
Variable x42 (boolean, object) has 50846 records (2.99%) with missing values
Variable x43 (boolean, object) has 50846 records (2.99%) with missing values
Variable x44 (boolean, object) has 50846 records (2.99%) with missing values
Variable x45 (boolean, object) has 50846 records (2.99%) with missing values
Variable x55 (boolean, object) has 50846 records (2.99%) with missing values
Variable x56 (boolean, object) has 50846 records (2.99%) with missing values
Variable x57 (boolean, object) has 50846 records (2.99%) with missing values
Variable x61 (content, object) has 32 records (0.00%) with missing values
Variable x62 (boolean, object) has 70978 records (4.18%) with missing values
Variable x63 (boolean, object) has 70978 records (4.18%) with missing values
Variable x64 (content, object) has 71061 records (4.18%) with missing values
Variable x65 (content, object) has 71061 records (4.18%) with missing values
Variable x71 (boolean, object) has 71061 records (4.18%) with missing values
Variable x72 (boolean, object) has 71061 records (4.18%) with missing values
Variable x73 (boolean, object) has 71061 records (4.18%) with missing values
Variable x74 (boolean, object) has 71061 records (4.18%) with missing values
Variable x75 (boolean, object) has 71061 records (4.18%) with missing values
Variable x85 (boolean, object) has 71061 records (4.18%) with missing values
Variable x86 (boolean, object) has 71061 records (4.18%) with missing values
Variable x87 (boolean, object) has 71061 records (4.18%) with missing values
Variable x91 (content, object) has 32 records (0.00%) with missing values
Variable x92 (boolean, object) has 140526 records (8.27%) with missing values
Variable x93 (boolean, object) has 140526 records (8.27%) with missing values
Variable x94 (content, object) has 140619 records (8.27%) with missing values
Variable x95 (content, object) has 140619 records (8.27%) with missing values
Variable x101 (boolean, object) has 140619 records (8.27%) with missing values
Variable x102 (boolean, object) has 140619 records (8.27%) with missing values
Variable x103 (boolean, object) has 140619 records (8.27%) with missing values
Variable x104 (boolean, object) has 140619 records (8.27%) with missing values
Variable x105 (boolean, object) has 140619 records (8.27%) with missing values
Variable x115 (boolean, object) has 140619 records (8.27%) with missing values
Variable x116 (boolean, object) has 140619 records (8.27%) with missing values
Variable x117 (boolean, object) has 140619 records (8.27%) with missing values
Variable x126 (boolean, object) has 32 records (0.00%) with missing values
Variable x127 (boolean, object) has 32 records (0.00%) with missing values
Variable x128 (boolean, object) has 32 records (0.00%) with missing values
Variable x129 (boolean, object) has 32 records (0.00%) with missing values
```

```
Variable x130 (boolean, object) has 32 records (0.00%) with missing values
Variable x140 (boolean, object) has 32 records (0.00%) with missing values
Variable x141 (boolean, object) has 32 records (0.00%) with missing values
Variable x142 (boolean, object) has 32 records (0.00%) with missing values
In total, there are 58 variables with missing values
```

Checking the cardinality of the int variables

Cardinality means the differents values of a variable, so we will see which feature will became dummy variables.

```
In [19]: for f in int_train_features:
             dist_values = int_train_features[f].value_counts().shape[0]
             print('Variable {} has {} distinct values'.format(f, dist_values))
```

```
Variable id has 1700000 distinct values
Variable x15 has 97 distinct values
Variable x17 has 119 distinct values
Variable x18 has 108 distinct values
Variable x22 has 498 distinct values
Variable x23 has 419 distinct values
Variable x27 has 193 distinct values
Variable x46 has 122 distinct values
Variable x48 has 167 distinct values
Variable x49 has 110 distinct values
Variable x53 has 499 distinct values
Variable x54 has 419 distinct values
Variable x58 has 149 distinct values
Variable x76 has 127 distinct values
Variable x78 has 184 distinct values
Variable x79 has 109 distinct values
Variable x83 has 498 distinct values
Variable x84 has 417 distinct values
Variable x88 has 141 distinct values
Variable x106 has 109 distinct values
Variable x108 has 123 distinct values
Variable x109 has 109 distinct values
Variable x113 has 500 distinct values
Variable x114 has 419 distinct values
Variable x118 has 159 distinct values
Variable x131 has 125 distinct values
Variable x133 has 158 distinct values
Variable x134 has 110 distinct values
Variable x138 has 500 distinct values
Variable x139 has 420 distinct values
Variable x143 has 493 distinct values
```

At this point i can't see if I will treat this variables as categorical and transform in dummy variables or treat them as quatitative variables.

1.2.5 Solution Statement

The solution proposed is to train a classifier to predict the labels of the extract OCR and the features created by the Tradeshift using the negative logarithm of the likelihood function to estimate the score for the classifier.

1.2.6 Benchmark Model

In this competition, the organizers give us fours different benchmarks scores: * The all zeros benchmark, where every label answer is zero; * The random benchmark, where every label was given a random value; * The halves benchmark, where every label was given a value of 0.5; * And finally the Tradeshift Baseline Benchmark.

The score of the four benchmarks is find below, in this evaluation metric lower is better:

Benchmark	Score
TS Baseline	0.0150548
All Halves	0.6931471
Random	1.0122952
All Zeros	1.1706929

1.2.7 Evaluation Metrics

The evaluation metric chosen by the organizers was the negative logarithm of the likelihood function averaged over Nt test samples and K labels. As shown by the following equation a + b =c. On the equation:

```
In [48]: %%latex
         $$\textrm{LogLoss} = \frac{1}{N_{t} \cdot K} \sum_{idx=1}^{N_{t} \cdot K} \textrm{LogL
         $$= \frac{1}{N_{t} \cdot K} \sum_{idx=1}^{N_{t} \cdot K} \left[ - y_{idx} \log(\hat{y
         $$= \frac{1}{N_{t} \cdot K} \sum_{i=1}^{N_{t}} \sum_{j=1}^K \left[ - y_{ij} \log(\hat
```

$$\textrm{LogLoss} = \frac{1}{N_{t} \cdot K} \sum_{idx=1}^{N_{t} \cdot K} \textrm{LogLoss}_{idx}$$

$$= \frac{1}{N_{t} \cdot K} \sum_{idx=1}^{N_{t} \cdot K} \left[- y_{idx} \log(\hat{y}_{idx}) - (1 - y_{idx}) \log(1 - \hat{y}_{idx})\right]$$

$$= \frac{1}{N_{t} \cdot K} \sum_{i=1}^{N_{t}} \sum_{j=1}^K \left[- y_{ij} \log(\hat{y}_{ij}) - (1 - y_{ij}) \log(1 - \hat{y}_{ij})\right]$$

- f is the prediction model
- θ is the parameter of the model
- $\hat{y}_{ij}$ is the predicted probability of the jth-label is true for the ith-sample
- log represents the natural logarithm
- $idx = (i-1) * K + j$

This function penalizes probabilities that are confident and wrong, in the worst case, prediction of true(1) for a false label (0) add infinity to the LogLoss function as $-log(0) = \infty$, which makes a total score infinity regardless of the others scores.

This metric is also symmetric in the sense than predicting 0.1 for a false (0) sample has the same penalty as predicting 0.9 for a positive sample (1). The value is bounded between zero and infinity, i.e. LogLoss $\in [0, \infty)$. The competition corresponds to a minimization problem where smaller metric values, LogLoss ~ 0, implies better prediction models. To avoid complication with infinity values the predictions are bounded to within the range $[10^{-15}, 1 - 10^{-15}]$

Example This is an example from the competition If the 'answer' file is:

```
id_label,pred
1_y1,1.0000
1_y2,0.0000
1_y3,0.0000
1_y4,0.0000
2_y1,0.0000
2_y2,1.0000
2_y3,0.0000
2_y4,1.0000
3_y1,0.0000
3_y2,0.0000
3_y3,1.0000
3_y4,0.0000
```

And the submission file is:

```
id_label,pred
1_y1,0.9000
1_y2,0.1000
1_y3,0.0000
1_y4,0.3000
2_y1,0.0300
2_y2,0.7000
2_y3,0.2000
2_y4,0.8500
3_y1,0.1900
3_y2,0.0000
3_y3,1.0000
3_y4,0.2700
```

the score is 0.1555 as shown by:

```
In [2]: %%latex
        $$L = - \frac{1}{12} \left [ log(0.9) + log(1-0.1) + log(1-0.0) +log(1-0.3) + log(1-0.0
```

$$L = -\frac{1}{12}\left[log(0.9) + log(1-0.1) + log(1-0.0) + log(1-0.3) + log(1-0.03) + log(0.7) + log(1-0.2) + log(0.8$$

1.2.8 Project Design

First of all, I will be preprocessing the data for the ingestion on the classifier, normalizing the numerical features, one-hot encoding the categorical features, convert of the content variables to a one-hot encoding and dealing with the invalids and missing entries. After the preprocessing, I will choose some ensemble methods to test the score and see what is the most important features from each ensemble method. Seeing what are the most important features, will train the same ensemble methods with some variation of the most impactful features like top 1%, 5%, 10%, 50% and see what is the quantity of feature has the most impact on the score. After that can apply a PCA method varying the number of features and train again the ensemble methods to see if the feature selection or PCA is the way to go. Passing this phase I can try to optimize the hyperparameter to the best-supervised learning method I find in the process above.

1.2.9 References

- [0] - https://www.kaggle.com/c/tradeshift-text-classification
- [1] - https://en.wikipedia.org/wiki/Optical_character_recognition
- [2] - http://ccis2k.org/iajit/PDF/vol.5,no.1/3-37.pdf - Zakaria Elberrichi, Abdelattif Rahmoun, and Mohamed Amine Bentaalah - Using WordNet for Text Categorization - The International Arab Journal of Information Technology, Vol. 5, No. 1, January 2008
- [3] - http://odur.let.rug.nl/vannoord/TextCat/textcat.pdf - William B. Cavnar and John M. Trenkle - N-Gram-Based Text Categorization - Environmental Research Institute of Michigan
- [4] - http://www.ijaiem.org/Volume2Issue3/IJAIEM-2013-03-13-025.pdf - Bhumika, Prof Sukhjit Singh Sehra, Prof Anand Nayyar - A REVIEW PAPER ON ALGORITHMS USED FOR TEXT CLASSIFICATION - International Journal of Application or Innovation in Engineering & Management (IJAIEM) - Volume 2, Issue 3, March 2013
- [5] - https://www.researchgate.net/publication/319688772_Using_text_mining_to_classify_research_pape - Sulova, Snezhana & Todoranova, Latinka & Penchev, Bonimir & Nacheva, Radka. (2017). Using text mining to classify research papers. 10.5593/SGEM2017/21/S07.083.
- [6] - https://www.researchgate.net/publication/280609521_Text_Classification_of_Technical_Papers_Base - Nguyen, Thien & Shirai, Kiyoaki. (2013). Text Classification of Technical Papers Based on Text Segmentation. 10.1007/978-3-642-38824-8_25.
- [7] - https://blog.tradeshift.com/hundreds-compete-to-improve-machine-learning-algorithm-for-5k-prize/
- [8] - https://www.kaggle.com/bertcarremans/data-preparation-exploration
- [9] - https://blog.metaflow.fr/ml-notes-why-the-log-likelihood-24f7b6c40f83
- [10] - https://quantivity.wordpress.com/2011/05/23/why-minimize-negative-log-likelihood/