COMP 488 - Machine Learning
Support Vector Machine & Gender Classification

Loyola University Chicago
Jose Luis Rodriguez

October 21, 2017

1 Overview

This report highlights the procedures to classify comments from the Humans of New York facebook page based on gender by first using two dictionaries to identify the gender in the comments them use Scikit learn to vectorize and standardize the data in order to perform cross validation, on an SVM and KNN classifiers. After the processing stage the dataset was split 70% (training) and 30% (testing) sets, the split would allows to measure the model performance (cross validation) and accuracy (F1 score).

- **Numpy:** A the fundamental package for scientific computing with Python.
- **Matplotlib:** A Python 2D plotting library which produces publication quality figures
- **Sckitlearn:** Simple and efficient tools for data mining and data analysis

1.1 Python Code

There is also a python (Version 3.6.3) code with the functions used on this report as well an implementation of the KNN classification algorithm. The code perform the analysis on the dataset and prints the results of the cross validation to the console and save charts with the performance of the models. It only uses the mentioned standard machine learning libraries, Numpy to standardize and manipulate data and Sckit-learn to create the train-test split, cross validation and svm model.

2 Model Feature Selection and Performance

The dataset used on this report is a collection of posts from the Humans of New York facebook page. This dataset contains, time of the post, source, link to the original post, comments, and metrics (likes count, comments count, share counts) for each post. The data processing step consisted of selecting just the features of interested such as the link to the original post to be able to trace back the post, also the comments column and all the metrics were kept. Two dictionary with words to identify gender were used to create a gender feature which classifies the post as male (0), female (1) and other (2) if the post contained a word or not in either of the dictionaries.

Table 1: Humans of New York - Dataset Shape (2149, 6)

message	**link**	**likes_count**	**comments_count**	**shares_count**	**gender**
Life settles you down.	... fb.com/humansofnewyork/...	6977	27	79	2
All that peace and love...	...fb.com/humansofnewyork/...	550	13	7	2
Boy giving a close-up...	...fb.com/humansofnewyork/...	1046	7	18	0
Furfect.	...fb.com/humansofnewyork/...	426	13	3	2
I loved all her grays.	...fb.com/humansofnewyork/...	364	7	5	1
⋮	⋮	⋮	⋮	⋮	⋮

Table.1 shows an snapshot of the data and per Figure.1 (appendix) we can see how the data is distributed between the two genders. From the plots we can infer that the captions referencing to women are more prevalent in the dataset as there are about 900 posts classified as male and 1200 as female. Its worth to note that the pictures (posts) captions are done by the photographer and sometimes refers to where the picture was taken or are quotes from whomever is in the picture.

3 Support Vector Machine – Classification Model

3.1 Feature Extraction and Selection

The table below shows the columns names as indexes that refer to the word used and their related count, the last two columns are the standardized values of likes, shares and comments in the dataset. With the proper features identified and standardized now the next step is to process the messages column which would give es the features (words) to classify the post by gender. To perform this task we can use the feature_extration module from scikit-learn to create a vector count of the words that are on the message column.

Table 2: Dataset – Standardize and Feature Selection

1	**2**	**3**	**4**	$\cdots$	**2696**	**2697**	**2698**	**2699**	**2700**	**2701**	**2702**
0	2	0	0	$\cdots$	0	0	0	0	0	-0.3558	-0.3724
0	0	0	0	$\cdots$	0	0	0	0	0	-0.2482	-0.2171
0	0	0	0	$\cdots$	0	0	0	0	0	-0.3154	-0.3105
0	1	0	0	$\cdots$	0	0	0	0	0	0.4115	0.5980
0	0	0	0	$\cdots$	0	0	0	0	0	-0.4259	-0.3881

After we do the feature extraction steps we should have a large sparse matrix of words now what we need to do is select the most relevant features from that sparse matrix in order to build our model which goal is to identify the gender of the post based on the given features. On this report two type of classification algorithms are used a LinearSVC and K-Nearest Neighbors, the next sections highlights each model and their performance.

Before taking a closer look to each model we can compare the models ability to classify the post by gender (performance) using the F1-score which can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

Table 3: Model Evaluation
Score Metric - Scale (0.0, 1.0) - (WORSE, BEST)

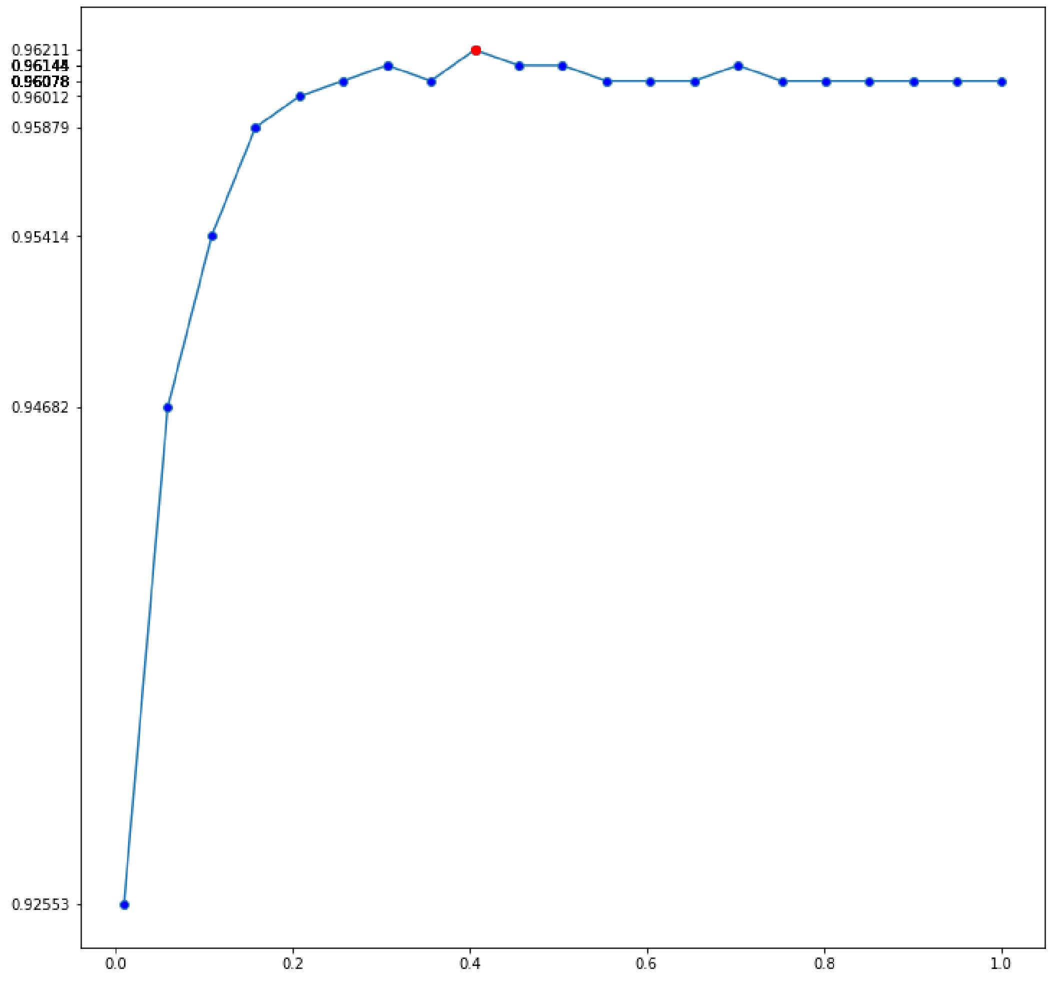
Model	**F1 Score**
Support Vector Machine	0.96211
K-Nearest Neighbors	0.74408

3.2 Performance and Cross Validation

The scikit-learn LinearSVC (Linear Support Vector Classification) as well as the cross validation implementation were used to create an validate the svm model. Initially the default values on the LinearSVC were used then using a 5 fold cross validation the regularization term was increased in each run.

In Figure.1 below the y-axis the cross validation mean score and on the x-axes is the value for the regularization term. We can see how as the regularization term increase the accuracy of the model improved with a maximum of 0.96 after that the model accuracy doesnt improve.

Figure 1: SVM Model – Performance



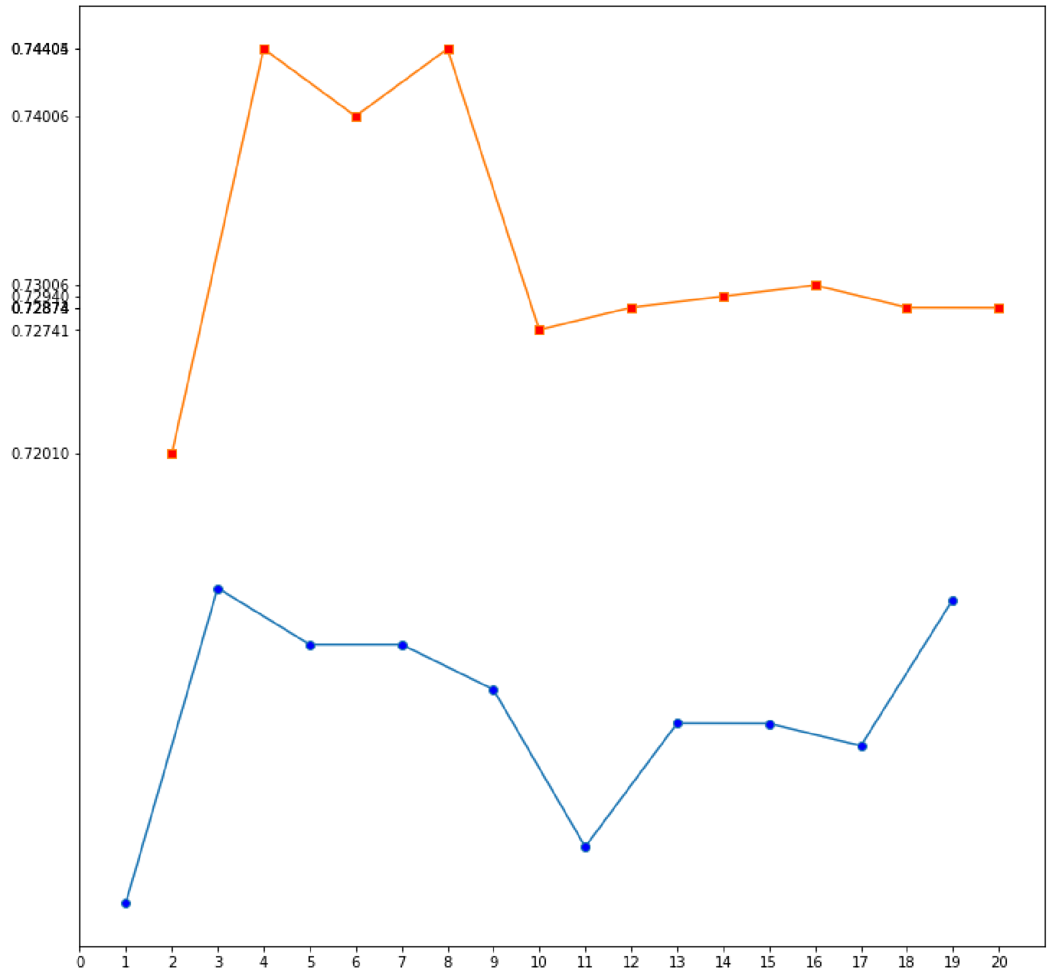
4 k-Nearest Neighbor – Classification Model

Performance and Cross Validation

The implementation for the KNN classifier model can be found in the appendix. A similar process than the svm classifier was follow in order to measure the performance of the classifier and cross validated the model. Due to the nature of the dataset the KNN model took a considerable longer time during the classification process.

In Figure.2 on the y-axis the cross validation mean score and on the x-axis is the value for k number of neighbors from 1 to 20. We can see an interesting difference between odd and even values of k with a maximum of 0.74 when $k = 4$, after that the model accuracy decrease.

Figure 2: KNN Model – Performance



5 Reference

Feature extraction 4.2.3. Text feature extraction – Scikit-Learn 0.19.0 Documentation, `http://scikit-learn.org/stable/modules/feature_extraction.html`

Preprocessing Data. 4.3.1 Standardization – Scikit-Learn 0.19.0 Documentation, `http://scikit-learn.org/stable/modules/preprocessing.html`

Support Vector Machines 1.4.1. Classification – Scikit-Learn 0.19.0 Documentation, `http://scikit-learn.org/stable/modules/svm.html#svm-classification`

Model evaluation. 3.3.2.8. Precision, recall and F-measures – Scikit-Learn 0.19.0 Documentation, `http://scikit-learn.org/stable/modules/model_evaluation.html`

6 Appendix

Figure 3: Dataset by Gender Metric Breakdown

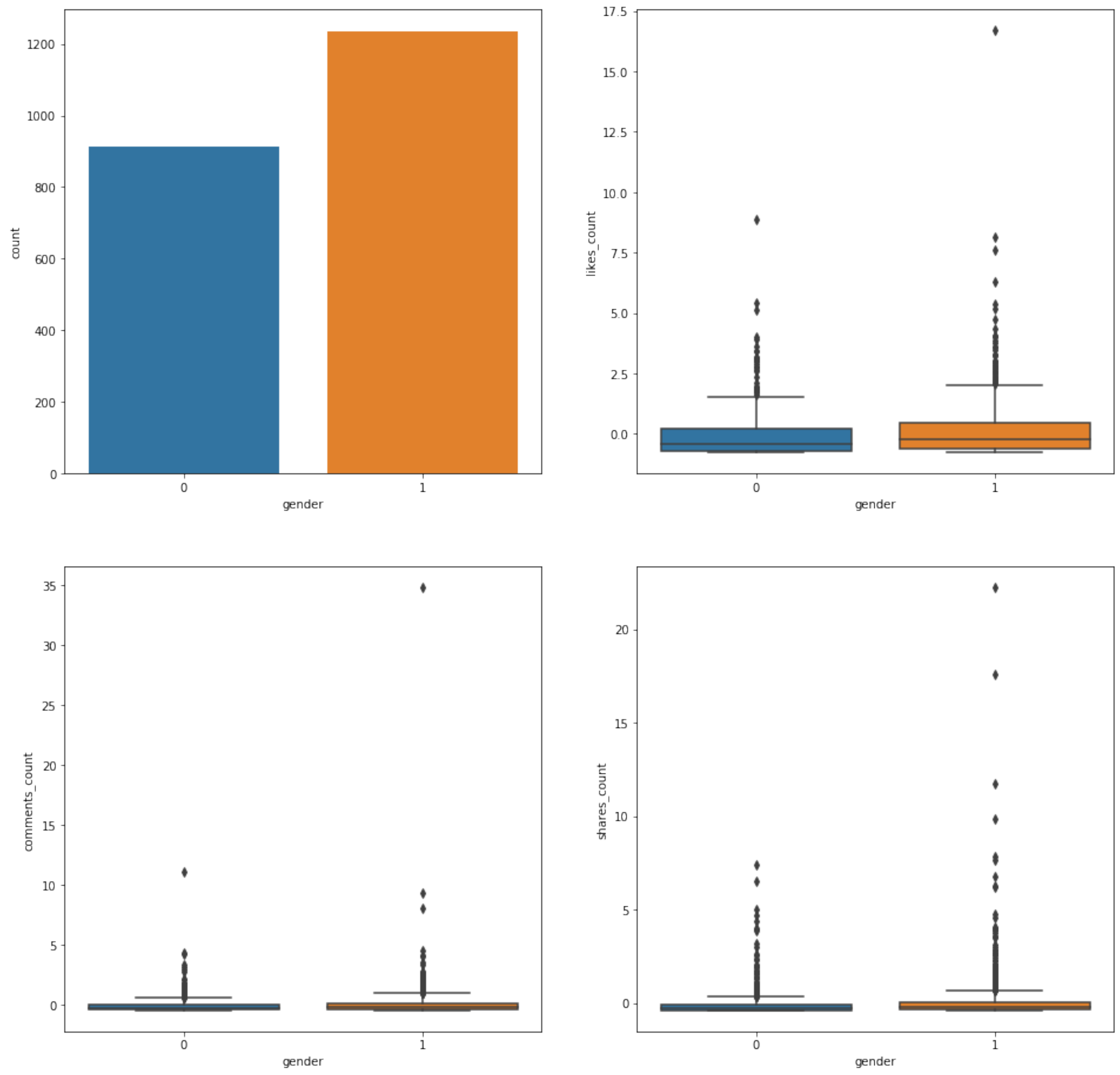


Figure 4: K-Nearest Neighbors Classifier – Implementation

```python
class KnnClassifier:

    def __init__(self, n_neighbors):
        self.n_neighbors = n_neighbors
        self._X = None
        self._y = None
        self.pred = None

    def fit(self, X, y):
        if self.n_neighbors > X.shape[0]:
            raise ValueError("K neighbors greater than Training sample")
        self._X = X
        self._y = y

    def predict(self, X):
        if X.shape[0] > self._X.shape[0]:
            raise ValueError("Test set (rows) greater than Train set")

        if X.shape[1] != self._X.shape[1]:
            raise ValueError("Test and Train sets have different shape")

        self.pred = np.empty(shape=(X.shape[0], 0) , dtype=self._y.dtype)

        for i in range(X.shape[0]):
            _test = X[i,:]
            _euclidean = []
            _class = []

            for n in range(self._X.shape[0]):
                dist = np.sqrt(np.sum(np.square(_test - self._X[n,:])))
                _euclidean.append(dist)

            idx = np.argsort(_euclidean)

            for k in range(self.n_neighbors):
                _class.append(self._y[idx[k]])
                self.pred = np.append(self.pred,
                                      Counter(_class).most_common(1)[0][0])
        return self.pred
```