Gender Prediction Using Character-Level Language Models

Andrew Cutler, Mehrnoosh Sarmashghi, Ali Siahkamari
{acut,msmshgi,siaa}@bu.edu

1. Task

Given a collection of someone's facebook statuses, can you predict their gender or personality? With a good language model (such as in the brain of the reader) natural language is rich with information about the author. However, given the number of colloquialisms and misspellings on Facebook, typical dictionary-based language models fail. We will implement a character-based CNN as well as an LSTM language model as feature extractors on the statuses. These will be followed by a fully connected classification network. We have the a five-dimensional personality vector (openness, conscientiousness, extraversion, agreeableness, neuroticism) for 2.5 million facebook users, as well as their gender. These labels will allow us to test performance of language model as well as explore benefits from domain adaptation.

2. Related Work

Author gender is closely related to sentimentality analysis, which has been richly explored. N-gram bag-of-words techniques, perform quite well and have long been the standard language model for these prediction tasks. However, these models discard syntactic information (eg. "threatening a child" vs "a threatening child") that is important to understand natural language. More recently, word and character level deep language models have been shown to perform better on a variety of text classification tasks, provided there are roughly a million training examples.

Neural networks have become increasingly popular for the task of NLP. Whereas feed-forward networks only exploit a fixed context length to predict the next word of a sequence, conceptually, standard recurrent neural networks can take into account all of the predecessor words. However, in a standard recurrent neural network, during the gradient back-propagation phase, the gradient signal can end up being multiplied a large number of times by the weight matrix. This means that, If the weights in this matrix are small, it can lead to a vanishing gradients. Conversely, if the weights in this matrix are large, it can lead to a situation called exploding gradients.

These problems are addressed by the Long Short-Term Memory neural network architecture which is capable of learning long-term dependencies, see[1]

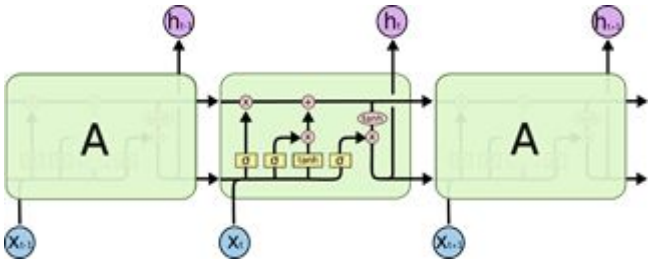

Figure 1. The repeating module in an LSTM

In this work, we exploit an LSTM language model on the Facebook statuses to predict gender and personality of people.

Yann Le Cun et al. argue that LSTM is a general model and require searching a larger parameter space[2]. Though there is little theoretical work to support this claim, empirical results are promising. Even with modest amounts of training data, they are able to come near N-gram performance.

Domain Adaptation and Multi-task learning are related topics that are still getting much attention for research. The problem of domain adaptation is when we are learning from a training examples from source set that its distribution is different from the target set. There could be different instances that we may have little or no training examples from the target domain. A summary on different classical approaches for domain adaptation is given in [3].

The multi task learning is a slightly different problem in which the target domain and the source domain are the same but there exist two different task (outputs). One trivial way to attack this problem is to train two different models for them separately. Past empirical work has shown that we get better performance if we try to learn these subjects simultaneously if the tasks are relevant in some sense [4, 5]. The setting of multi-task learning is suitable for our problem. We want to train two classifiers on the same data one for gender

classification and another one for personality prediction.

3. Approach

We will construct two simple models (LSTM and CNN) we believe will perform reasonably well, then add features like resnet gates, batch normalization, and dropout as time allows. Once we get a working model we will be able to experiment with parameters and test initialization schemes. Our first model will have six convolutional followed layers by three fully connected layers. We will use dropout, batch normalization, and max pooling implemented from the Keras library over tensorflow. Much of the code for a simple model is available online, and we will likely follow a tutorial [7]. There is less support for state of the art techniques such as character res-nets and we will code them up ourselves.

After training the initial model we will be able to make better decisions about pre-processing, domain adaptation, and goals for more sophisticated models. The upper bound on model complexity we will strive for is Yan LeCunn's 2017 proposed deep CNN. This uses all previous techniques listed as well as residual gates that allow training up to 49 layers.

4. Dataset and Metric

The mypersonality app is a third-party facebook quiz taken by 7.5million people. Over 2 million also elected to offer all information on their facebook profile (statuses, pictures, relationship status, friends list) to be used for research. Andrew has obtained access to the database. We will use statuses from individuals that have labels in both gender and personality. Our metric will be classification error for gender and mean absolute/squared error for personality. We will compare LSTM, CNN and their respective multi-target learning schemes.

We anticipate spending some time on preprocessing the data. The corpus is enormous, and the one-hot coding scheme the CNN requires for input further inflates the data. However, having some experience with this dataset, we believe we will be able to do these considerations won't detract from our efforts making a model.

4.1 Data Preprocessing

The mypersonality dataset consists of status updates from roughly 3 million individuals. The raw csv file is 25 million lines long. However, the collection is plagued by missing data and those statuses must be filtered out to use in a supervised learning setting. Statuses from a single author are concatenated until they reach 1024 characters. After cleaning and packeting the data we are still left with a sizeable set of 1.4 million text chunks all bounded by 1024 characters with labels for both gender and personality. The character set is limited to 60 distinct characters: numbers, lowercase latin letters, punctuation, a special character to denote separation of statuses within a packet, and an end of packet character repeated until the length of the group of statuses is 1024.

5. Results

5.1 Baseline Shallow Methods

We have implemented some basic methods such as Naive Bayes and SVM on a bag of character. This approach is not very common and mostly people use a bag of words model. The reason that we used a bag of character model is that in our study it turned out that females tended to use more punctuation letters and more smiley faces than males. So we wanted to make sure that the deep models are learning some higher level concepts than just counting the punctuation letters.

Naive Bayes:

What naive bayes does is assuming independence of the different features conditioned on the class label. So this assumption reduces the number of parameters needed to model a discrete distribution from exponential to linear in terms of features. The graphical model of this model is in figure(2)

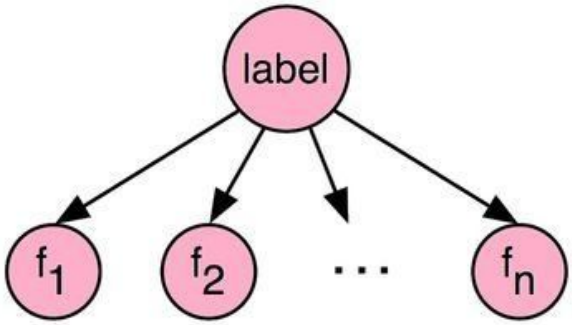

Figure 2. The graphical model for a naive bayes model

In our problem features are the number of occurrences of different characters and the likelihood model is simply a multinomial distribution over the occurrences of different characters as bellow:

$$p(y \mid x_1, \ldots, x_{60}) = \varphi_y \prod_{i=1}^{60} \varphi_i^{x_i}$$

In which x_i is the number of occurrences of the i^{th} character in the status chunk and φ_i is the corresponding probability of occurrence.

After training this model we got a accuracy of %50 percent on a equalized data set of females and males which is the same is random guessing. Then we did a backward feature selection and got to accuracy of %62. fig(3). it turned out that the worst character to be used as a feature was the character “space” because people use space very prevalently but it doesn’t have anything to do with gender classification. Although we have a very good estimate of probability of occurrence for space, even if it’s a little bit different between females and males, when prediction contributes to a large error. Lastly turned out that only using the four characters “:”, “|”, “y” and “0” would give us an accuracy of 60 percent. The justification for the character “:” is that it comes a lot with smiley faces and females tend to use it more often. For character “|” it's important because it's the status separator and its corresponding feature counts the number of statuses in a status chunk. The reason for the two other characters are not that clear.

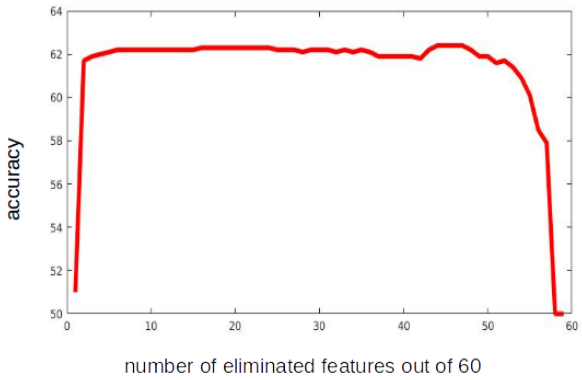

Figure 3. Accuracy Vs number of eliminated features in backward feature selection

Kernel SVM:

The basic idea of a linear svm is to separate a linear separable data via a line with the largest margin. obviously this is not a very robust way and also not practical on common datasets because almost nothing is linearly separable. So there is two more trick to end up with a very powerful kernel SVM. The first is to allow some points to be on the wrong side of the line if that helps a lot on increasing the margin. The second is when rewriting the optimization formulas for SVM problem it turns out that it could be written in term of only inner-products of different data points. Although this transfers the problem from the order of number of features to order of number of data points, it allows to use more powerful feature representations without explicitly having to know the representation. We implemented kernel SVM both on MNIST and our dataset (the features are still the number of occurrences of the characters) with different kernels. The result of the SVM is in the table (**). The RBF kernel didn’t do as well as the linear kernel and that is just because we didn’t put very effort on finding the best hyperparameters for them since this was not the focus of our project.

Data Set	Kernel	Train Acc	Test Acc
MNIST	linear	%98	%95
MNIST	RBF	%94	%92
Gender Data	linear	%72	%62
Gender Data	quadratic	%70	%62

Table 1. Prediction Accuracy Using SVM

5.2 Char-level RNN

For character level RNN, first, we have used LSTM on MNIST data to see how our model performs. For MNIST data, we have an image with 28*28 pixels. We consider each row of pixels as an input. So we would have 28 sequences of inputs. We apply LSTM with 128 hidden units in Tensorflow on MNIST . After 100,000 iterations, we got %98.4 accuracy.

For our dataset, the models would be like the following diagram.

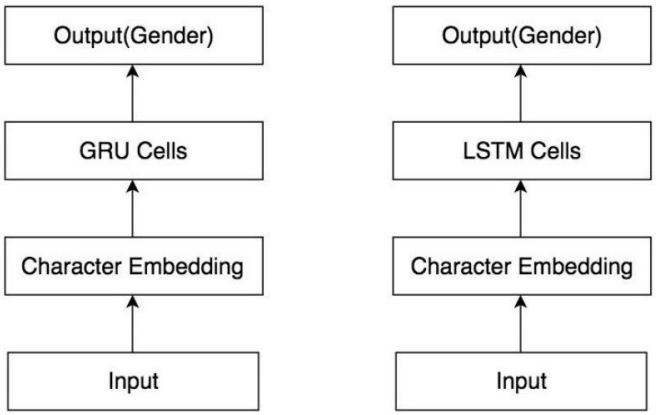

Figure 4. Architectures of LSTM and GRU implementation

The size of Input is 2 (millions) * 1024, so each example includes 1024 characters. For converting inputs characters to vector of numbers, we have used character embedding instead of using one-hot vector. If we use one-hot vector, each character would be converted to 60 dimensions (because we have 60 different characters in our dictionary). However, by using character embedding, each character is converted to 16 dimensions vector. This method is more efficient that one-hot vector. So our input is altered to 2(millions)*16*1024.
Then, we feed data to LSTM cells. For computing outputs of LSTM we use the following equations:

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$
$$\widehat{C}_t = tanh(W_c.[h_{t-1}, x_t] + b_c)$$
$$f_t = \sigma(W_f.[h_{t-1}, t] + b_f)$$
$$C_t = f_t * C_{t-1} + i_t * \widehat{C}_t$$
$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

Our model includes 200 hidden units and the number time steps are 1024. The batch size is 128. First, we run the model but it takes a lot of time to run, so instead of computing all of the equations, in Tensorflow there exists more efficient method. After using this method, we got %80 accuracy after 8 epochs by spending less time.
We were curious to see the difference between LSTM and GRU. Therefore, we implement GRU on the data set. After training, we get %81 accuracy by spending less time. We can conclude that GRU performs better than LSTM at least on our data.
We implement multilayer LSTM and try for a couple of times to run two layers LSTM but unfortunately, our job was canceled on SCC. And also we implement LSTM by using dropout but again we could not run it on SCC. However, both of these models work on other data set like MNIST.

Data set	Model	Test Acc
MNIST	LSTM	%98
MNIST	GRU	%99
Gender Data	LSTM	%80
Gender Data	GRU	%81

Table 2. Accuracies of LSTM and GRU

5.3 Char-level CNN

The Offbit Blog [6] provided the skeleton for our first model. Their task was slightly different; reviews were broken into sentences which were processed by CNN's. These higher-level sentence representations were then passed through a bi-directional LSTM. However, we gained a lot by looking at their embedding approach and general use of the Tensor Flow wrapper Keras. As seen in the table below, even a simple model does well above the most naive model (guessing female every time) which gets 66%.

Earlier this year, Le Cun introduced a Very Deep CNN (VDCNN) that we were able to build from scratch and test data on. As seen in the figures from the figures below [2], there is a convolutional block that consists of two batch normalized convolutional layers. Each character is first embedded in a low-dimensional space (following Le Cunn, we chose 16), then filtered through a series of these convolutional blocks. After two such blocks, there is a pooling layer that decreases the length by ½. The number of filters doubles after each such pooling, as has shown to be effective in computer vision.

In practice, models ought to be judged on performance and cumbersomeness, with a penalty for deeper models on the latter metric. However, as this is a deep learning course we made both depth and accuracy a goal and continued training deeper even when we saw no performance boosts. This served as a learning exercise on training methods as well as a sanity check on the discipline. As Table 3 shows, our best CNN is nine layers deep. The following models of 35 and 75 layers took considerably longer to train, required more tuning of hyperparameters, and were generally less stable. If we had access to a larger dataset the extra work may have payed off with modest gains in performance rather than more degrees of freedom to overfit training data.

Model	Depth	Accuracy
CNN (dropout)	3	0.75
CNN (dropout)	5	0.77
CNN (batchnorm)	9	0.83

CNN (batchnorm)	35	0.83
CNN (batchnorm)	75	0.81

Table 3. Accuracies for CNN

Note: All convolutional and recurrent networks are followed by one and two fully connected layers respectively. As a baseline accuracy, 66% of the dataset is female.

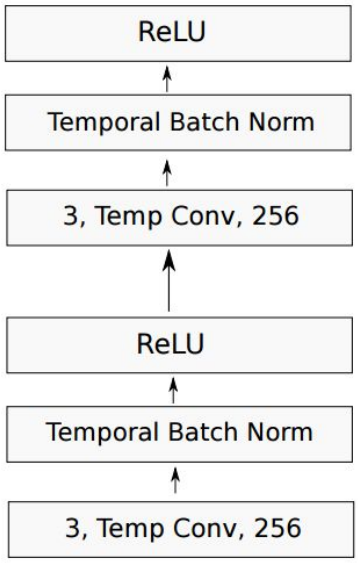

Figure 5. Convolutional block of the VDCNN

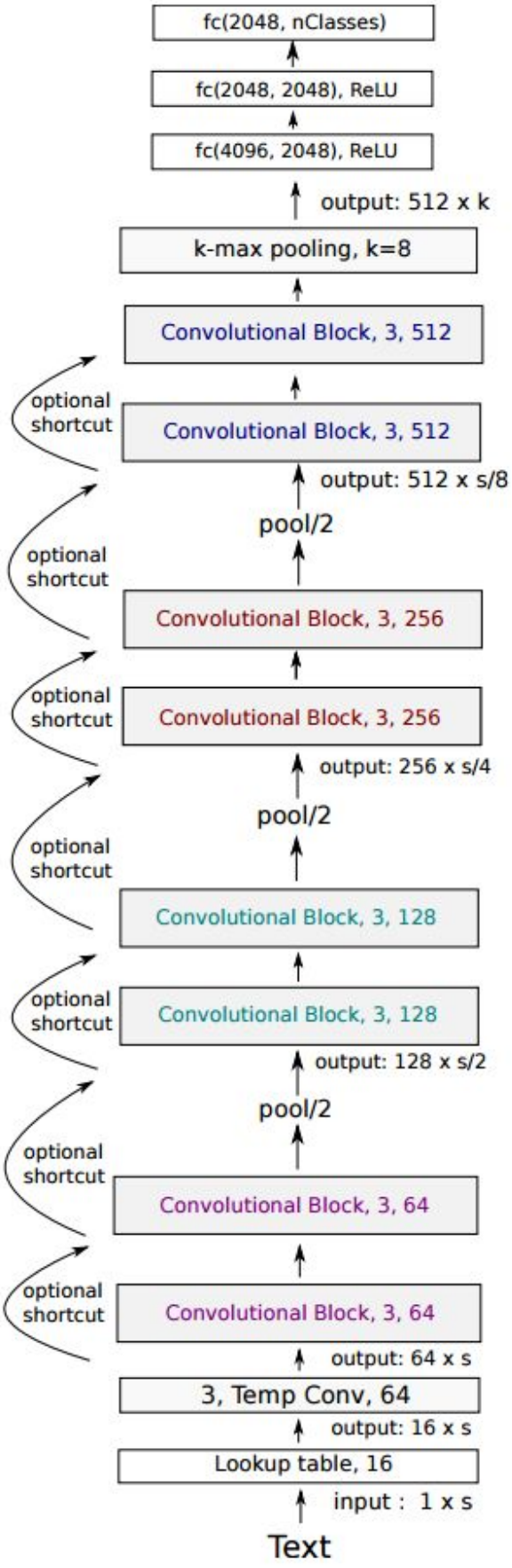

Figure 6. Very Deep CNN architecture

Ada boosted CNN

Ada-boost is a special case of a more general family of models called the multiplicative weight update algorithm with an exponential loss. The basic idea behind boosting is to use multiple weak classifiers to come up with a better classifier. The first classifier is trained as you would normally train any specific classifier. For training the latter classifiers the algorithm is told to put more emphasis on the points that previous classifiers have gotten wrong. And finally a weighted majority vote is taken from all the classifiers as the output of the algorithm. The weight of vote of different classifiers is related to how well they did on the weighted version of the training set that they were trained on. The specific formulas could be found in [8].

In our problem the weak learner was set to a three layer CNN network followed by a fully connected layer with dropout of 0.7 on every layer. Then we defined Id's for each training example along with a weight for them. And we fed these weights to the tensorflow trainer so that it could use a weighted sum of cross entropy losses when training. And notice that there is no need for each batch to have the total weight as of other batches.

After training with Ada-boost actually we found that the accuracy of the weighted sum is the same as the accuracy of the first classifier. This happened because the vote of the first classifier was more important than the sum of votes of all the other classifiers. The same thing happened every time we trained these networks. Although it seems paradoxical with the guarantee of the Ada-boost to end up with total training accuracy of 0 it is not the case. Because as we checked classifiers after the 5th step of the boosting, they were all like a coin flip and could not satisfy the requirement of the guarantee.

The results of the Ada-boost is in table(4)

Iteration	1	2	3	4	5
Confidence	1.23	0.13	0.34	0.18	0.0
Weighted error	%22	%46	%41	%45	%50
Training error	%22	%22	%22	%22	%22
Test error	%23	%23	%23	%23	%23

Table 4. Boosting results

Word-Based model

We tried to implement our model on a bag of words model, because words makes sense more than characters when using classical methods. We wrote a code to find words and make a dictionary based on them. But because people usually do not use words properly (for example sometimes they do not use space between the words or because of misspelling), we got a lots of different words which some of them do not make sense at all. We got 70,000 words for the first one million characters. And since this has needed a complicated parsing problem we didn't have time to implement it.

6. Detailed Roles

Task	**File names**	**Who**
Naive Bayes Implementation	Naive_bayes folder	Ali
Kernel SVM implementation	SVM folder	Ali
CNN Adaboost	CNN_boosting folder	Ali
LSTM	LSTM folder	Mehrnoosh
GRU	GRU folder	Mehrnoosh
Word-Based model	Char2Word	Mehrnoosh
Data Preprocessing	Processing folder	Andrew
Deep CNN	vdcnn_a.py	Andrew

Table 5. Detailed roles

7. Code repository

https://drive.google.com/open?id=0B0VrbH9B6XizeFhfQ3dwNng2Rlk Discription of processing files in the google drive "information" tab.

References

Zhang, Xiang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." *Advances in neural information processing systems*. 2015.

1) S. Hochreiter and J. Schmidhuber, Long short term memory. Neural computation, 9(8):1735–1780,1997.
2) Le Cun, Yan, et al. "Very Deep Convolutional Networks for Text Classification." *arXiv preprint arXiv:1606.01781* (2016).
3) Jiang, Jing. "A literature survey on domain adaptation of statistical classifiers." *URL:*

http://sifaka. cs. uiuc. edu/jiang4/domainadaptation/survey 3 (2008).

4) Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." *IEEE Transactions on knowledge and data engineering* 22.10 (2010): 1345-1359.

5) B. Bakker and T. Heskes. Task clustering and gating for Bayesian multi–task learning. Journal of Machine Learning Research, 4: 83–99, 2003.

6) “How to read: Character Level Deep Learning”. *Offbit*. N.p.,2017. Web. 28 Feb. 2017

7) Five Factor Model of Personality: Goldberg LR, et al. (2006) The international personality item pool and the future of public-domain personality measures. J Res Pers 40(1):84–96.

8) Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. Sp ringer, Berlin: Springer series in statistics, 2001.

9) Jozefowicz, Rafal, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures." *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015.