Capstone project: Dialogue classification

Moritz Bach

September 12th, 2018

I. Definition

Project Overview

Natural language provides interesting challenges in the of application of machine learning techniques. Examples of this are (i) sentiment analysis, for example in the context of movie and product reviews, and (ii) text classification. The latter is a problem that is very present in our daily lives, as our incoming email is algorithmically categorized into spam or no-spam. Correct classification can mean the difference between receiving unsolicited junk mail in your inbox and having important messages gone missing, due to it being erroneously tagged as spam.

In this project I will train text classifiers to predict the speaker of a dialogue line from the iconic TV show "Seinfeld" (NBC, 1989-1998). The data to be used comes from a complete collection of dialogue scripts from the show, which is provided by seinology.com and which was scraped using the code from a GitHub repository (https://github.com/amanthedorkknight/the-seinfeld-chronicles).

Problem Statement

The goal of the project is to find the machine learning technique that performs best at answering the question “Who said that?”. In predicting the character that speaks a certain dialogue line, the project deals with a multi-class text classification problem.

In order to train multiple text classifiers on the show scripts, a bag-of-words (BOW) approach will be employed. This transforms the feature space from a single column of natural text into vectors of word frequencies, thereby losing punctuation and word order.

The results generated in this project will be the performances of the different classifiers, as measured by a chosen metric. To improve the results, two strategies will be used: (i) feature space transformation techniques will be employed and their effects on classifier performance will be measured. (ii) Hyper-parameters of the classifiers will be optimized using a grid search approach.

The desired solution to this problem would be a classifier that performs significantly better than chance in predicting the speaker of a given dialogue line. Finding a solution like that would indicate that characters in the show Seinfeld have distinct archetypical vocabularies.

Metrics

In this multi-class problem, we are only interested in correctly classifying as many dialogue lines as possible. When misclassifying, we do not differentiate between wrong classes. Hence, we are simply interested in the fraction of correctly classified samples, which is exactly given by accuracy:

$$accuracy = \frac{number\ of\ correctly\ classified\ samples}{sample\ size}$$

Accuracy can take values between 0 and 1, where higher values mean a better performance in correctly classifying samples.

II. Analysis

Data Exploration

Dialogue scripts from the show are provided by seinology.com and were scraped, using the code from a GitHub repository (https://github.com/amanthedorkknight/the-seinfeld-chronicles). The data is comprised of six columns (Tab. 1) and counts 54,616 rows.

Table 1: Columns of the Seinfeld dialogues dataset

Column name	Data type
ID	numeric
Character	text
Dialogue	text
EpisodeNo	numeric
SEID	text
Season	numeric

Table 2: Sample rows from the dataset

ID	Character	Dialogue	EpisodeNo	SEID	Season
16	JERRY	I told you about Laura, the girl I met in Michigan?	1	S01E01	1
17	GEORGE	No, you didnt!	1	S01E01	1
308	ELAINE	Wel-Well, I just dont appreciate these little courtesy responses, like Im selling you aluminum siding.	1	S01E01	1
575	GEORGE	(to Jerry) Ready?	2	S01E02	1
616	KRAMER	Uh, Jer, well you know, I was cookin and I, I uh, I came in to get this spatula...and I left the door open, cause I was gonna bring the spatula right back!	2	S01E02	1
3174	[Setting	Jerry's apartment]	8	S02E08	2
5063	(knock on door	this is Jack and Doris, neigboors)	3	S03E03	3

The ‘Dialogue’ column contains natural language and includes spelling mistakes, colloquialisms and sounds that the characters make(Tab. 2). Furthermore, the dataset does not purely contain dialogue lines, but also setting information (indicated by brackets) and stage directions (indicated by parentheses). The latter can be embedded in a dialogue line or be split across both ‘Character’ and ‘Dialogue’ columns.

In this project, only the columns 'Character' and 'Dialogue' will be used, as labels and features respectively. The feature space will be transformed, from natural language to vectors of word frequencies. Dialogue lines from non-recurring characters will be grouped into one class ‘Other’ (for more see section III Methodology: Data preprocessing).

Exploratory Visualizations[1]

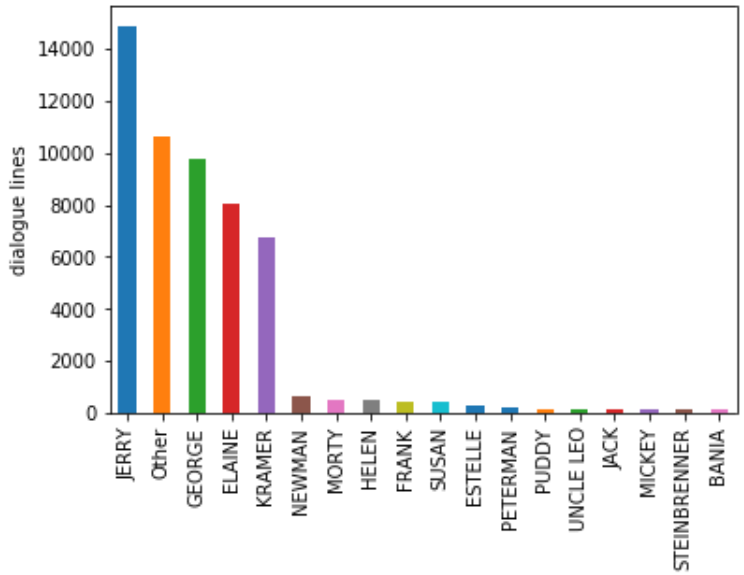

Fig. 1: Absolute frequency of dialogue lines per character; preprocessed data

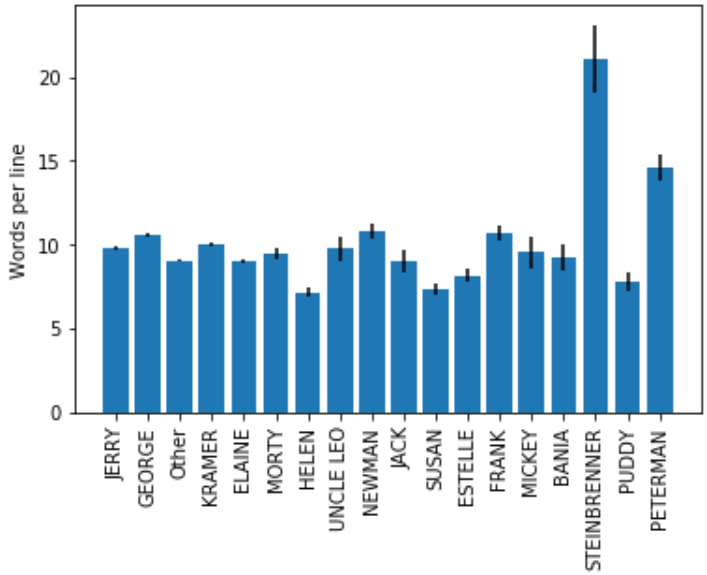

Fig. 2: Sentence length per Character (mean +SE); preprocessed data

The dialogues in the show are dominated by the four main characters Jerry, George, Elaine and Kramer (Fig. 1). Jerry Seinfeld has by far the most dialogue lines. The combined total of all dialogue lines spoken by minor parts exceeds that of the character George. The show also features thirteen recurring secondary characters, which all have well below 1000 dialogue lines.

The number of words per dialogue line is mostly around 10 for each character (Fig. 2), while three female characters (Helen, Susan, Estelle) and the auto mechanic Puddy speak 7-8 words per line on average. Only the characters of Mr. Peterman and Mr. Steinbrenner consistently have longer lines (on average 15 and 21 words, respectively).

[1] Section refers to preprocessed data (see Methodology: Data Preprocessing)

Algorithms and Techniques

Bag of words (BOW) approach

In order for any classifiers to work on the dataset it is necessary to transform the feature space from natural language to word count vectors. This is similar to one-hot encoding for categorical data.

All entries in the 'Dialogue' column are split into vectors of so-called n-grams, where n denotes an integer for the number of words to be coupled. Then, a dictionary of all n-grams is learned and the occurrence of every n-gram is scored for each row. If the dataset consisted only of the two rows (ID 16,17 from Tab. 2), then the transformed feature space would look like (Tab. 3) (using 1-grams). Information about word order and punctuation is lost in this operation.

Table 3: Hypothetical word counts after feature space transformation

ID	Character	i	told	you	about	laura	the	girl	met	in	michigan	no	didnt
16	JERRY	2	1	1	1	1	1	1	1	1	1	0	0
17	GEORGE	0	0	1	0	0	0	0	0	0	0	1	1

Text vectorization options

In addition to applying the basic BOW approach, the effect of these vectorization options will be investigated:

- Specification of ngram_range:
  In addition to the default setting of only using 1-grams (ngram_range=(1,1)), the effect of incorporation of 2-grams will be investigated. By using a combination of 1- and 2-grams (ngram_range=(1,2)), as well as using 2-grams exclusively (ngram_range=(2,2)). As an example, converting George's dialogue line (Tab. 2) to 2-grams would result in "no you" and "you didnt".
- Stop word removal:
  The most common words in a document usually don't carry much information content. Examples of this are "the", "is", "at", "which", and "on". These so-called stop words will be removed, using a pre-defined list of English stop words, thereby decreasing the feature space.
- Stemming:
  Stemming is the practice of reducing inflected words to their word stem, in order to reduce the feature space and group words of similar basic meaning. An example would be the conversion of "looks", "looked" and "looking" to "look".
- Term frequency–inverse document frequency (TF/IDF):
  TF/IDF is a weighting statistic, that gives weights to the words vectors, rather than just using word counts. It favors terms that are frequent in a dialogue line (term frequency), but scarce over all dialogue lines (inverse document frequency). It is very popular to use this in search and recommender engines, however it can also be applied to classification (Yoo & Yang, 2015)[2].

[2] Yoo, J. Y., & Yang, D. (2015). Classification scheme of unstructured text document using TF-IDF and naive bayes classifier. *Advanced Science and Technology Letters*, *111*(50), 263-266. http://onlinepresent.org/proceedings/vol111_2015/50.pdf

Text classification algorithms

To predict character from dialogue lines, four distinct types of classification models will be trained:

- Multinomial Naïve Bayes

  Naïve Bayes classifiers are a natural choice for text classification problems. They employ a purely probabilistic method to word frequencies and are very efficient with large datasets and input spaces. Naïve Bayes classifiers are a very popular baseline method for text classification, but unbalances classes can reduce their performance (Frank & Brouckaert, 2006)[3].

- linear Support Vector Machines (SVM)

  Linear SVMs were chosen because they are good at dealing with a highly dimensional input space and they can perform well on sparse data (Joachims, 1998)[4]. Also, most text problems are linearly separable, which makes linear SVMs a great candidate as they maximize the margin around the separating hyperplane.

- Random forests

  Random forests are chosen as an example for ensemble methods (decision tree ensembles). Decision trees are often used for classification problems, however one of the downsides is a habit to overfit on the training set. Random Forests try to correct for this.

- Artifical neural networks (ANNs)
  Neural networks were chosen as the most elaborate classifier. A lot of research is being done in regards to them. Neural networks are best known for their great performance on image classification, but they can also work well on text classification[5].

Benchmark

Two types of benchmarks will be used to grade the performance of the text classifiers in this project:

Benchmark I:

Because of the large class imbalance (Fig. 1) the first benchmark is just the fraction:

$$\frac{number\ of\ lines\ from\ character\ Jerry}{total\ number\ of\ dialogue\ lines} = \frac{14894}{53795} = 0{,}2769$$

This is equal to the chance of scoring a correct prediction, when always predicting the label to be 'JERRY'. This should serve as the absolute baseline for any classifier.

---

[3] Frank, E., & Bouckaert, R. R. (2006, September). Naive bayes for text classification with unbalanced classes. In *European Conference on Principles of Data Mining and Knowledge Discovery* (pp. 503-510). Springer, Berlin, Heidelberg. https://link.springer.com/content/pdf/10.1007/11871637_49.pdf

[4] Joachims T. (1998) Text categorization with Support Vector Machines: Learning with many relevant features. In: Nédellec C., Rouveirol C. (eds) Machine Learning: ECML-98. ECML 1998. Lecture Notes in Computer Science (Lecture Notes in Artificial Intelligence), vol 1398. Springer, Berlin, Heidelberg https://link.springer.com/content/pdf/10.1007%2FBFb0026683.pdf

[5] https://www.opencodez.com/python/text-classification-using-keras.htm

Benchmark II:

The second benchmark will be the result of the multinomial Naïve Bayes classifier (hyperparameter values: alpha=0.85, fit_prior=True, class_prior=None) trained on data transformed without further vectorization techniques (use only 1-grams, no stop word removal, no stemming and no TF/IDF). The Naïve Bayes classifier is computationally very efficient and has historically been applied to many text classification problems. Thus, is should serve as a good benchmark for the other classifiers.

III. Methodology

Data Preprocessing

Some preprocessing steps had to be carried out, to clean up the dataset and achieve higher consistency in labels.

- All labels 'MAN', 'WOMAN' were changed to 'Other'.
- To remove label inconsistency of dialogue lines of the character Uncle Leo, all labels from 'LEO' were merged to those with 'UNCLE LEO'.
- Rows containing setting information (start with "[" in column 'Character') and stage directions (start with "(" in column 'Character') were removed
- Stage directions within labels and dialogue lines were removed, using the regular expressions '\(.*\)' and ' *\(.*\)'.
- For every label that occurred just once, these rows were deleted.
- All remaining labels of characters with a total number of dialogue lines less than 100 were changed to 'Other'

After this, the transformation of the feature space from natural language to word frequency vectors was done (Bag of Words approach, for more see section II Analysis: Algorithms and Techniques).

Implementation

The project was implemented in Python (Version 3.5.5, http://www.python.org) in a Jupyter notebook (http://jupyter.org/). For all text classifiers, implemenations from the package scikit-learn (Version 0.19.1, http://scikit-learn.org/stable/) were used. While the construction and training of the artificial neural networks (ANNs) relied on implementations from the package keras (Version 2.1.6, http://keras.io) using a TensorFlow backend (Version 1.8.0, https://www.tensorflow.org/).

The work on this project followed these steps:

1. Loading of dataset
2. Cleaning and preprocessing of data
3. Data exploration
4. Training of classifiers with scikit-learn
5. Creation of training of ANNs (see section 'Refinement')
6. Generation of confusion map of the best performing classifier

The training of the scikit-learn classifiers was implemented in the function 'run_treatments' and will be explained in more detail. A list of 'treatments' was defined which represent the different combinations of vectorization techniques (see section 'Algorithms and Techniques'). The treatments were looped and in every iteration the function 'run_treatments' was called and the results were saved for later analysis. The first part of the function incorporated the logic that determines how the features spaces is supposed to be transformed, given the supplied treatment.

The second part of the function then executes the whole process of training and testing the three classifiers (see section 'Algorithms and Techniques'). It includes the following steps:

- Transformation of the feature space according to supplied treatment
- Execution of a stratified, randomized split into training and testing data (labels and features)
- Definition of hyperparameter values to be used during grid search, for each classifier
- Iterate over list of classifiers, train classifier on training data using grid search
- Apply optimized classifier to predict labels on test data
- Return training and testing accuracy as result, along with hyperparameter value.

The stratified data split was chosen because of the large class imbalances, to make sure that all classes are represented equally in training and testing set, relative to their sample size. During grid search, the training data was split into three cross validation sets, to increase generalizability.

Refinement

In the beginning, the removal of rows with the conglomerate class 'Other' (see section 'Data preprocessing') was thought to increase accuracy. However, the result was the opposite. Because of that and the fact that it would have been an arbitrary decision anyway, it was not further investigated (results are not shown).

Since the treatments were conceptualized from the start of the project, the remaining measure to increase classifier accuracy was the optimization of hyperparameter values using grid search (Tab. 4).

Table 4: List of tested parameter values during grid search for the respective hyperparameters of each classifier

Classifier	hyperparameter	Tested values
MultinomialNB	alpha	1, 0.85, 0.7, 0.5, 0.3
LinearSVC	C	1, 10, 100, 1000
RandomForestClassifier	n_estimators	10, 50, 90, 150, 250, 350
	min_samples_split	2, 20, 30, 50, 60, 90, 120
	min_samples_leaf	1, 5, 20

The MultinomialNB classifier was the only one which showed different optimal values for its hyperparameter, depending on treatment. For the LinearSVC classifier, the optimal hyperparameter value was always C=1. After initial testing the hyperparameters for the RandomForestClassifier, were kept constant with optimal values at n_estimators=250, min_samples_split=90, min_samples_leaf=1.

In regards to neural networks, a number of simple network architectures have been tested (Tab. 5). All of these networks used 'relu' activation units, except for the output layer, which always featured 'softmax' activation. The optimization metric was accuracy (loss='categorical_crossentropy', optimizer='adam').

Table 5: List of trained neural network architectures, with FC=fully connected layer, DO=dropout layer and number of nodes, dropout rate respectively

Model	Architecture
NN1	FC 100 → DO 0.1→ FC 100 → DO 0.1→ FC 18
NN2	FC 100 → DO 0.1→ FC 100 → DO 0.1→ FC 100 → DO 0.1→ FC 18
NN3	FC 100 → DO 0.3→ FC 100 → DO 0.3→ FC 100 → DO 0.3→ FC 18
NN4	**FC 100 → DO 0.5→ FC 100 → DO 0.5→ FC 18**
NN5	FC 200 → DO 0.5→ FC 200 → DO 0.5→ FC 18
NN6	FC 512 → DO 0.5→ FC 512 → DO 0.5→ FC 18
NN7	FC 512 → DO 0.3→ FC 256 → DO 0.3→ FC 128 → DO 0.3→ FC 18
NN8	FC 256 → DO 0.3→ FC 256 → DO 0.3→ FC 128 → DO 0.3→ FC 128 → DO 0.3→ FC 18

The model 'NN4' (Tab. 5) performed better than other any other tested network with more layers or different number of nodes or dropout rates.

IV. Results

Model Evaluation and Validation

The final results for all classifiers showed accuracies greater (or equal to) 0.3 (Tab. 6, 7). This can be considered quite low. All further vectorization techniques decreased classifier performance (Tab 6, not shown for other classifiers). This was unexpected. The multinomial Naïve Bayes classifier showed the highest accuracy on the testing data. The best performing neural network performed the worst compared to the other types of classifiers (Tab 7). These results were unexpected.

Table 6: Treatment of data transformation and its effect on classification accuracy for the multinomialNB classifier (optimized hyperparameter value given)

N-gram range	Stop word removal	Stemming	TF/IDF	Classifier	Test acc.	Difference to benchmarks	
						BM I	BM II
(1, 1)	False	False	False	MultinomialNB 'alpha': 0.85	0.3268	18%	-
(1, 1)	True	False	False	MultinomialNB 'alpha': 0.7	0.3204	16%	-2%
(1, 1)	True	True	False	MultinomialNB 'alpha': 1	0.3195	15%	-2%
(1, 1)	True	True	True	MultinomialNB 'alpha': 0.7	0.3136	13%	-4%
(1, 2)	**False**	**False**	**False**	**MultinomialNB** 'alpha': 0.7	**0.3341**	**21%**	**2%**
(2, 2)	False	False	False	MultinomialNB 'alpha': 1	0.3134	13%	-4%

Table 7: Comparison of prediction accuracy by text classification algorithms

N-gram range	Stop word removal	Stemming	TF/ IDF	Classifier	Test acc.	Difference to benchmarks	
						BM I	BM II
(1, 2)	False	False	False	MultinomialNB 'alpha': 0.7	0.3341	**21%**	**2%**
(1, 1)	False	False	False	RandomForestClassifier 'n_estimators': 250, 'min_samples_split': 90	0.3187	15%	-2%
(1, 1)	False	False	False	LinearSVC 'C': 1	0.3098	12%	-5%
(1, 1)	False	False	False	Neural network (NN4, Tab. 5)	0.3079	11%	-6%

For the sake of model generalizability and robustness, two actions were taken. First, the data was partitioned into training and testing data using a randomized stratified split. This was supposed to guarantee that all classes are represented both training and testing sets, relative to their sample size. Second, the training data was split into three cross-validation sets for hyperparameter optimization while the testing data was held out. Still, some stochastic effects could be observed, when performing the initial split with a different random generator seed. Different splits would cause small differences in classifier accuracy (up to 2%) and sometimes result in different optimal hyperparameter values (only multinomial NB).

Despite that, the results can be considered robust in terms of the effect of the feature space transformations and classifier type.

Justification

All models performed better than Benchmark I (at least 11% increase in accuracy), although only marginally (Tab 6, 7). The performance of the models was well below expectations, which will be further discussed in sections Reflection and Improvement.

Results from all models are consistent in the fact that additional feature space transformations (stop word removal, stemming and TF/IDF) decreased classification accuracy. Combining these vectorization techniques further decreased accuracy. As stop word removal and stemming reduce the feature space, it seems that usable information was lost. Why the different word weighting from the application of TF/IDF results in lower accuracy is unclear.

The multinomial Naïve Bayes classifier was set as a benchmark because of its simplicity and long history of use in text classification problems. The expectation was that the other classifiers would perform better on the data, due to their higher complexity. However, this was not the case. The most complex classifier tested here (neural networks) performed the worst on the data.

The difference between Benchmark II and the model with the highest accuracy is very small (2%). The performance increase makes intuitive sense, given that the superior model had more information available (included the use of 2-grams).

V. Conclusion

Free-form Visualization

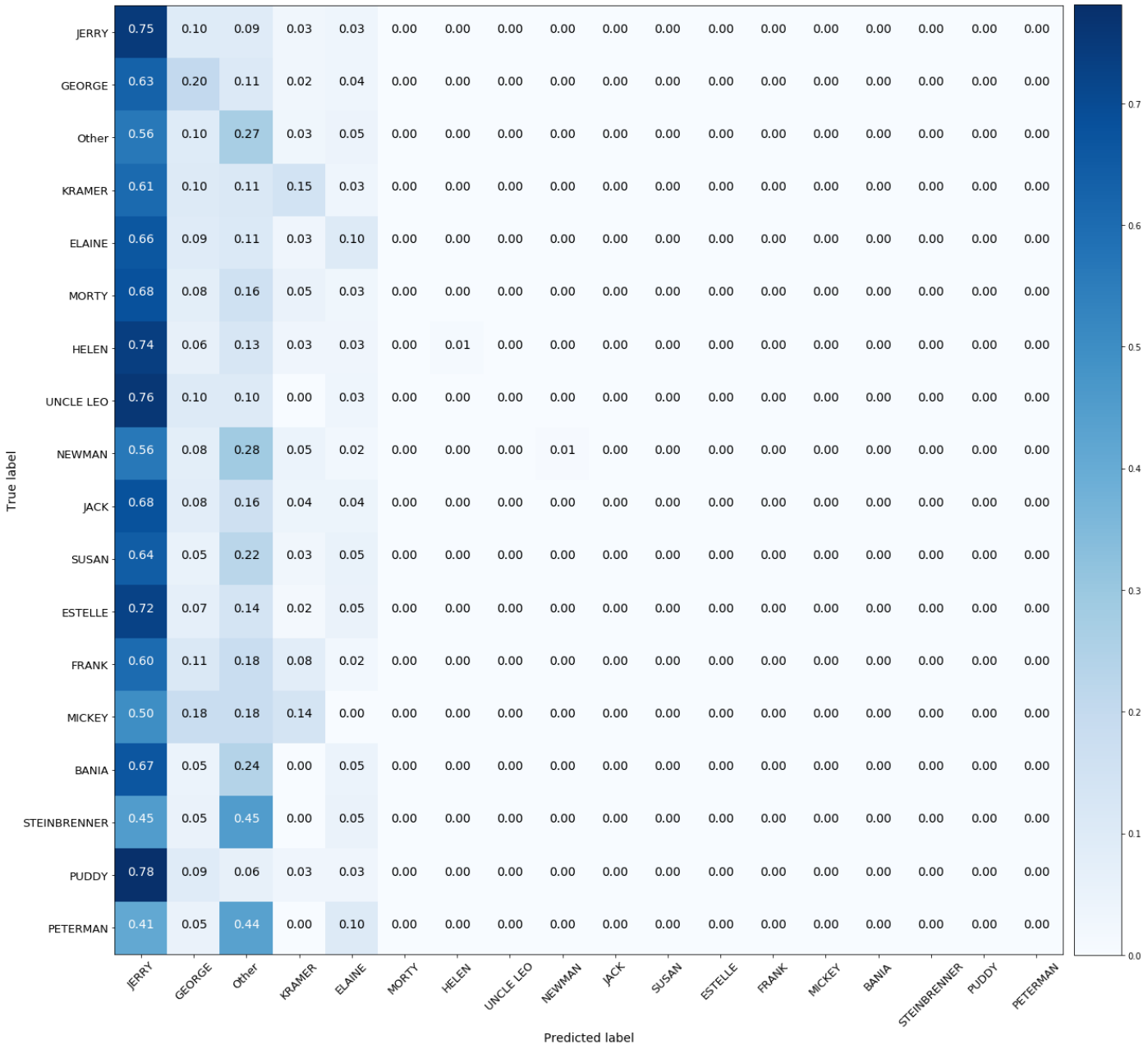

Fig. 3: Confusion map of best performing classifier (MultinomialNB (Tab. 7))

Fig. 3 shows the accuracy of the best classification model on a per-class level. Aside from four main characters and the conglomerate class “Other”, all characters are correctly predicted below 2% of the time, mostly below 1%. Regarding the main characters, the predictions are accurate only 10-20% of the time for George, Elain and Kramer. The highest class-specific accuracy is achieved for the character Jerry (0.75). For any class, the predicted label is Jerry in 41-76% of all its dialogue lines.

This figure shows the influence of the large imbalance of sample size per class (Fig. 1). The predictions are highly biased towards the most frequent class, while classes with very few samples are almost never predicted.

Reflection

To summarize, this project was about the classification of dialogue lines from the TV show Seinfeld. I started to work on the project by first exploring the data. Then I cleaned and preprocessed the data. The dialogue lines were converted from natural text to bag of words data, using different vectorization techniques. Then, I trained four different types of classifiers (Naïve Bayes, linear SVM, RandomForest & ANN) on the data and tried to optimize their performance.

What was interesting to me in the project was that the data was not perfect and that I had to clean it a bit. Another interesting aspect was that the fact that it was a multi-class classification problem with very unbalanced sample sizes. For these two reasons I think it was closer to a real-world problem, compared to previous projects of the Nanodegree.

The most difficult part of the project was to improve the accuracy of the classifiers. The additional vectorization techniques actually decreased the performance and hyperparameter optimization using grid search only marginally improved the accuracy. That is why the results were well below my expectations.

In terms of show content, there are a few recurring themes that are distinct for the characters. For instance, George is famous for lying a lot, Kramer often comes up with foolish business ideas and tries to sue large companies numerous times and Jerry regularly breaks up with women for superficial reasons. None of these themes are likely to be captured by text classification models.

The fact that the prediction accuracy of the classifiers is for the main characters is so low (Fig. 3) suggests that the dialogue lines do not differ very much and that the characters do not have archetypical vocabulary.

In regards of the data, one important aspect is the large imbalance of the classes, as this creates a bias towards the prediction of the character Jerry. The almost none-existent correct predictions for the minor characters is probably due to the small number of samples (Fig. 3).

By applying the bag of words approach, information is immediately lost in terms of word order, sentence structure and punctuation. Additionally, the machine learning algorithms have no access to context and all those other things that make a character: its actions and the delivery of the lines from the actor through the means of tonality, body language and facial expressions.

In the end, the problem of classifying show characters by dialogue lines just doesn't seem suitable for machine learning algorithms.

Improvement

One of the biggest drawbacks of my execution of this project might be considered the number and simplicity of the tested neural network models. More complex neural network architectures could have been employed and might have performed better than my best-performing model. However, the training time even for the simple models was too long on my machine, to do more elaborate experimentation.

Also, random forests and neural network models did show a lot of over-fitting. Accuracy on the training data was sometimes as high as 90%, however these models did not generalize and performed worse on the test data than the Naïve Bayes classifier. Therefore, it can be hypothesized that training more complex networks or training for higher number of epochs, would also only lead to more overfitting.

Besides training neural networks with only fully connected layers, I could have explored the use of convolutional neural networks (CNNs). I briefly experimented with that, but training time was even

higher than for my networks with purely fully connected layers. Again, it can be hypothesized that CNNs would also overfit and might not improve accuracy on the testing set.

It seems to be a problem of the data and the whole approach, rather than of specific classifiers.

Therefore, a possible improvement on character classification might come from a completely different approach. Instead of applying the bag of words approach, it might be interesting to use word embeddings and do sentiment analysis on these by training neural networks. The learned sentiments could then be used to train a classifier (i.e. SVM) in order to predict the character.

Using sentiments, rather than a bag of words, would be an abstraction to a higher level. A character might display similar moods and motives over the course of the show, while he might use different vocabulary and phrasing. Therefore, sentiments might be more recurring and thus better suited for machine learning.

While this would be interesting to try, it is purely speculative if this would actually improve the classification accuracy. After all, it is not clear if there are any distinct differences between the characters in that regard.