Test assignment

by Artem Glazunov

I. Theory

1. What are the main problems of modern NLP and NLU?

I will try to briefly describe some of the main problems, from my point of view:

- Data may exist in the form of tables, graphics, notations, page breaks, etc., which need to be appropriately processed for the machine to derive meanings in the same way a human would approach interpreting text.
- It is hard to comprehend the context of text. As far as I know, none of the existing models exhibit "real" understanding of natural language.
- Another problem is capturing emotions, which are one of the keys to "real" NLU.
- And also, a key question is, what biases and structure should we build explicitly into our models to get closer to NLU. On the other hand, for reinforcement learning, many researchers claim, that we would ultimately want the model to learn everything by itself, including the algorithm, features, and predictions.
- Another problem of our future – creating the universal language model, which can enable people to read news, that was not written in their language, ask questions about their health, when they don't have access to a doctor, etc.
- In the end, I would mention the problem of quality measurement. In other words, how can we adequately measure, how well our systems perform on new, previously unseen inputs? Or, how do we adequately measure, how well our systems generalize? After all, how should we modify our models, so that they generalize better?

2. Which libraries would you pick to use for the following cases and why (all problems should be solved for the Russian) - Sentiment analysis - Multi-label classification - Dependency parsing - POS-tagging - NER

General libs (useful more or less for all tasks):

- numpy, pandas (everybody loves matrices and tables)
- re
- sklearn.model_selection (cross_val_score, Repeated Stratified K-Fold GridSearchCV)
- I would also try Bayesian optimization of hyperparameters (Hyperopt lib), because of less amount of iteration due to the usage of previous steps.
- sklearn.pipeline (Pipeline)
- matplotlib, seaborn (visualizations)
- plotly (interactive charts)
- flask (for presentation of the model)
- I also use Google Colab for many tasks, because of its good free features – 16 GB RAM, GPU, TPU support.

Sentiment analysis

For parsing:

- Scrapy (one of my favorites), bs4, requests (for “from scratch” purpose)

For preprocessing:

- bs4 (delete html)
- num2words (number to words)
- nltk (stopwords, to delete)
- pymystem3 (fast lemmatization)
- natasha (Segmenter, MorphVocab, NewsEmbedding,NewsMorphTagger, NewsNERTagger, NamesExtractor, AddrExtractor, Doc)

It contains wrappers for many modules out-of-box, that’s why I like it.

- pymorphy2 (MorphAnalyzer) (substitution for lemmatization)
- imblearn (oversampling, undersampling)

Bag of words approach:

- sklearn.feature_extraction.text (CountVectorizer, TfidfVectorizer)
- Personally, I have not tried more advanced approaches for sentiment analysis, but I would also try Gensim with its Word2Vec, FastText and Doc2Vec because of using mutual links between words, and that fact can become crucial for quality, I guess.

Creating ML model:

For BOW I was taught to use linear algorithms, because of large dimension of sparse matrix:

- sklearn.naive_bayes (MultinomialNB)
- sklearn.linear_model (LogisticRegression, SGDClassifier)
- sklearn.svm (LinearSVC)

The best results for my tasks was shown by LinearSVC and LogisticRegression.

- Vowpal Wabbit (Great realization for SGD, but they also have more advanced solvers. Very good for big amounts of data (GBs), also fits for online algorithm due to SGD. It worked for me in “bag of sites” binary task on Kaggle, but, frankly speaking, I have not used it for SA yet)
- Important to admit, that using probability models is good, because the tasks are usually imbalanced, and we want to see the probabilities, and then choose the best threshold for our task, especially in the case, when there are different costs of mistakes for classes. That’s why sklearn.calibration.CalibratedClassifierCV is a very useful thing to implement it, if the choice is not the LogisticRegression algorithm, but it is comparatively slow due to CV.
- It is important to note, that W2V and D2V approaches allow us to use RandomForest, XGB, LGBM due to less dimensions of the feature space, and, I suppose, nobody would argue, that these algorithms are very impressive sometimes.
- Instead of W2V we can use BigARTM to get vectorized text representation with topic probability interpretation. Alternatively, we can use LDA for dimension

reduction using topics. LDA has a very good representation in genism library. Gensim also has lots of visualization variants and metrics.
- I would also try RNN someday, but in order to get significant benefits, I guess, we have to gain lots of data for appropriate training of the model.
I want to mention also Dostoevsky lib (RegexTokenizer for tokenization and pretrained FastTextSocialNetworkModel). In my experiments, I've slightly touched it, but, in my opinion, this FastTextSocialNetworkModel is good only for social media analysis due to its origin.

Multi-label classification
- sklearn.neighbors.KNeighborsClassifier, sklearn.naive_bayes.MultinomialNB (as a baseline)
- LinearSVC, LogisticRegression (I guess, the main choice for BOW for me yet),
- XGB, LGBM, RandomForest (with maybe LDA, BigARTM W2V and D2V),
Keras, TensorFlow (with more data and much more time) – it may give the best result, provided with the mentioned conditions.

Dependency parsing
- natasha (Doc, NewsEmbedding, Segmenter, NewsMorphTagger NewsSyntaxParser)
- Using this library, we can get syntax structure of sentences and create visualization for dependency trees with ease.

POS-tagging
- natasha (for words - NewsMorphTagger,for text also would use– Doc, Segmenter, NewsEmbedding),
- pymorphy2 (MorphAnalyzer) (substitution)
Using these libs, we can get parts of speech for words and use it with ease for lemmatization purpose, for instance.

NER
- natasha (Doc, NewsEmbedding, Segmenter, NewsNERTagger, also useful - MorphVocab, NamesExtractor)
- NewsNERTagger helps us for NER and NamesExtractor may be helpful for extracting last names, for instance.

- deeppavlov (loads very big DL model (~2 GB) and uses configs, build_model to create the model)
I tried it, but it seems to me, that such a big size is slightly inconvenient, although its DL origin may give better quality for our solution of finding named entities task.

3. How would you evaluate a classification model, which metrics would you use?

In this section, I will describe my typical SA model. I am not a super professional in this field yet (and that is why I am here), and my experience may seem funny to you, but it wouldn't be fair to use some other kind of experience from top kagglers, for example.
So, first and foremost, I would examine our corpus, print out some texts and estimate a balance of classes in our data (in other words, EDA). The second step would be the main preprocessing pipeline (described in the next section). Then, I would split data into train and validation parts, after that - use my baseline approach – CountVectorizer and SGD logreg with accuracy metric (previously model.predict (data_val)) for balanced classes. For imbalanced case, I would use AUC ROC score (previously predict_proba(data_val)), that is, roughly, the probability of the right order for mapping class 1 label in a random pair of objects from classes 1 and 0. Then, I would also try lemmatization (for Russian corpus) or stemming (for English task) and oversampling strategies (random, smote and others) and maybe undersampling.
The second part of the process is creating better model. I would create a pipeline with data_transformer and classifier in it (def model_pipeline(data, transformer, classifier), for example), use cross_val_score with 5 folds (maybe with RepeatedStratifiedKFold cv strategy, 3 repeats, for less variance and better stability of cv results) for choosing good transformer-classifier pair. Then I would use GridSearchCV with the same strategy for tuning the hyperparameters for my model (regularization, n_gramms range, min and max words frequencies, and so on). And again, for balanced task I would use simple accuracy score, but for imbalanced classes, I would prefer AUC ROC score. It is important to note, that simply relying on mean cv results is not the best strategy, because it can be misleading due to outliers. That's why, before choosing the best algorithm, I would explore all cv results for all folds.
The last part of model creation for me is choosing the best threshold for our imbalanced task (if it's such). Firstly, I would get classification report and plot precision and recall scores for a range of thresholds. Then, I would find the best threshold, maximizing f1 score or simply using our plot (it depends on the business objective of our model, we can maximize only recall, but it would cause bad accuracy of triggering, so, we should find the most balanced solution).

3. Main pipeline for the text pre-processing

In general, my preprocessing algorithm is:

- (optional) Extract, normalize and delete (optional) named entities;

- (optional) transform emoji into words;

- clean the text from artifacts (html tags, special symbols) and punctuation;

– convert to lowercase and lemmatize;

– delete stop words;

- convert numbers into text;

- delete digits;

- (optional) add extracted and normalized named entities in the end (it can be important, when it comes to searching for negative information about certain persons and texts clustering).

I also have some code with an example in the attached files *(Typical_preprocessing_and_architectures_comparison.html)*

5. Microservices or monoliths? Why.

Here is a small table with structured features of both architectures.

Microservices	**Monoliths**
More reliable Can be implemented via distributed nodes. If some nodes fail, the other still work. Also, the philosophy of Microservices, as far as I remember, is like “every part must do one thing, and do it well”	*Simpler* Have less parts, easy to manage, although it isn’t good for big projects for debugging, I guess.
Easier Scaling with less resources Allow to scale only the necessary components.	*Lower Latency* Calls between components are faster
Less Communication By breaking a big team into many smaller communication channels can be reduced.	*Higher throughput* Don’t have the overhead of images and orchestration. For non-distributed workloads, where absolute performance matters, Monoliths are better.
More Agile Rapid deployment, convenient scaling and efficient communication are good for business.	-

So, for scalable modern projects with many teams, Microservices are the best choice, I guess. But it's important to note, that a big project can start as a monolith, because of its simplicity and higher speed for small projects.

In addition, in the attached files I have a small example of the architecture comparison (Typical_preprocessing_and_architectures_comparison.html). Don't judge me harshly.)

6. Describe the hardest programming task you've been facing with. It's not necessarily ML task, could be just a programming. Why this task was hard to accomplish? What was your solution for the task? Can you share a github project?

In this section, I would describe my educational project on predicting 102 time series of averaged taxi trips counts for 102 zones of New York. The task was solved on Google Colab, without payable servers. So, this task was divided into several steps.

1. Firstly, I had to download minimum 15 GBs of raw data and create efficient aggregation code with memory cleaning. It's taught me, how to avoid memory leakage and to use gc.collect() properly. I want to note, that overall, through this algorithm I've aggregated more than 200 Gb of data during my feature engineering process, without memory leakage.

2. Then, I had to implement a clustering of given time series, tune parameters of autoregressive SARIMAX model with exogenous features (to delete week seasonal effect) for each cluster, and, after all, fit SARIMA model for each of 102 zones.

For clustering, I used a simple approach of extraction only April 2016 data from time series, scaled and clustered those short data. Having it done, I used Agglomerative Clustering algorithm on the data (without time warping transformation, because I tried it, and it was not good due to importance of saving time shifts between series) and came to 5 clusters with comparatively good metrics. Using more clusters was bad, because optimization of SARIMAX parameters is not the simplest process. I also used Folium to look at all my cluster, all was normal from logical point of view (clusters represented different geo zones, for example airports, downtown areas and outskirts).

Then, I found centers of the clusters and tuned parameters of SARIMAX models with exogenous features for the centers.

The tricky part was, when I was creating a wrapper for SARIMAX model. The thing is, that ordinary ARIMA with exogenous features uses algorithm of multiple optimization iterations to find the best regression coefficients, but I figured out, that for this task it was too much. That is why, I divided SARIMAX and Linear Regression model for week seasonal component, wrapped it into convenient class

and used additional interesting trick. The thing is, that in my case it wasn't necessary to fit the model on the whole dataset, which would be the task for several days on a server. I simply fitted the model on 1 starting month, and then "extended" models via Kalman filter approach on the whole period. I also used Yei-Johnson transformation to stabilize dispersion. It turned out, that this approach had very little drop in quality. So, instead of many days of calculation, I managed to solve the task in an hour.

3. When the SARIMAX approach had been implemented, I started to use regression algorithms. After feature engineering and attempts to solve the task of prediction in 1 to 6 hours perspective using 6 regression xgb models, I managed to increase the quality via 612 model, 6 for each zone. So, after all my attempts, I came to comparatively good results (in both quality and performance), despite the fact, that I did not use much data, only 6 months.

This is the link to github:

https://github.com/art-glazunov/NY_TAXI_trips_ts_prediction

For convenience, here and further, all the notebooks are also in html format, for quick reviews.

7. Did you work with VCS? Which one?

I studied GIT for a while some time ago and can work with it, but due to the fact, that I haven't worked in a team with many versions of files, I don't have much practice with this. So, I am still fascinated by rebase, cherry-picking and conflicts resolving staff, and also need a week or two to refresh knowledge and train skills.

8. Did you work with Github Actions?

Only studied a bit, read a few articles and watched some videos on it. For me, it's cool to implement CI/CD directly on Github.

9. How familiar are you with Docker and other orchestration tools?

I studied it a bit, but haven't implemented. For me, the idea of working on or deploying projects, using isolated containers with appropriate environment, seems very cool, especially for the microservices architecture, where on one server many teams can work in different environments. I cannot say much of orchestration tools, except the base – they help manage many containers on servers, help automate the maintenance of applications, able to replace failed containers

automatically, and manage the rollout of updates and reconfigurations of those containers during their lifecycle. For example, there are Kubernetes and Docker Swarm tools.

10.What is ed25519 and why is it concerning to be better than ecdsa?

Ed25519 is the eddsa signature scheme using SHA-2 and Curve25519.
Eddsa (Edwards-curve Digital Signature Algorithm), in public-key cryptography, is a digital signature algorithm based on performance-optimized elliptic curves, such as Curve25519 and Curve448-Goldilocks. Such algorithms, for example, can serve as a means to create authentication and authorization systems for accessing to servers.
SHA-2 (Secure Hash Algorithm 2) - is a set of cryptographic hash functions.
Ecdsa (Elliptic Curve Digital Signature Algorithm) - offers a variant of the Digital Signature Algorithm which uses elliptic curve cryptography.
Ed25519 is simpler than ecdsa, easier to understand and to implement. It also usually offers better security than ecdsa. In addition, Ed25519 algorithm is faster. Its main strengths are its speed, its constant-time run time (and resistance against side-channel attacks), and its lack of nebulous hard-coded constants. In addition, I've heard about some political concerns about ecdsa (related to backdoors in software, if I'm not mistaken). So, ed25519 is considered to be better for most modern apps.

11. Do you have any experience in data mining?

Yes, on my current work, as an analyst for a big company, I often explore many sources of data to make recommendations for the management and create ranking models, I am also creating a prototype of an anomaly detection model now. Some of my tasks were related to searching for a negative harmful information in mass media as well.
Moreover, I have completed 4 educational ML projects, such as:
- Parsing for data and sentiment analysis of mobile phones reviews https://github.com/art-glazunov/Santiment-analysis-project;
- Predicting a big amount of time series with complicated seasonal component using autoregression and regression approaches (mentioned in the 6-th answer) https://github.com/art-glazunov/NY_TAXI_trips_ts_prediction;

- Churn prediction for a telecom operator https://github.com/art-glazunov/CHURN_prediction_telecom_operator;
- User identification via her/he site session history https://github.com/art-glazunov/User_identification.

During each of the projects, I was aggregating some statistics from raw data and looking for hidden mechanisms and dependences, that could help me predict the right class label or build a regression model efficiently.

I want to mention, that:

- In the Sentiment analysis project, I've applied Scrapy framework to collect thousands of reviews from different sites (for example, https://otzovik.com/), this led me to a result in top-16% in a Kaggle in-class competition.
- During the Churn prediction project, I've managed to get 4th result in Kaggle in-class competition, applying different realizations of the gradient boosting algorithm (Catboost, lgbm, xgb), as well as feature engineering and feature selection technics.

2. Practice

The project is divided into EDA, preprocessing, model selection, final solution and testing parts. It also includes my model for preprocessing with useful functions. All steps of the project you can find in the ipynb files.

See also html files for a quick review.

I only want to mention, that I've tried LSA (Latent semantic analysis) and LDA (Latent Dirichlet allocation) models, and used small stratified data for model selection purpose, and the whole prepared dataset for the final solution. As metric, the coherence was used. I also tried to visualize and print all useful information. In the end, I've selected and trained the following model:

```
gensim.models.LdaMulticore(corpus=gensim_corpus,
                           id2word=gensim_dict,
                           num_topics=20,
                           random_state=100,
                           workers = 3,
                           passes=20,
                           eta = 0.005,
                           alpha = 0.007
                           )
```

You can find the fitted model in the results2 folder.