Project documentation

Machine learning Fall 2019

Jozef Kubík

In this document I will try to cover results of my project, describe used method, write about different problems encountered during coding, show some interesting things etc.

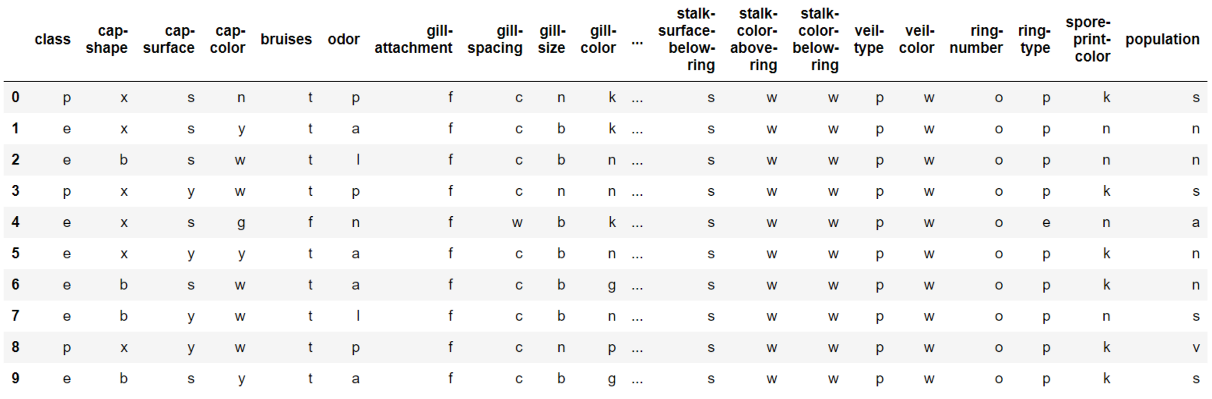
1. Introduction

This project was aimed to show some different classification methods and their results when used on specific dataset. Dataset chosen for this project is called „Mushroom classification" and it's available on Kaggle: https://www.kaggle.com/uciml/mushroom-classification .

2. Dataset

Dataset and its structure are nicely described on the website mentioned above, and I refer to it as a main point if reader is interested in content of dataset and better description of its data and data values meaning. Not everything is well documented here or the project code; in case of any question about data, I suggest you go check there.

Dataset contains 8124 rows and 23 columns of data. That means 8124 unique mushrooms with 23 features each. Feature of each mushroom describes some part of it, e.g. cap, odor, color and most importantly edibility. This most important feature is called "class" and it's our target variable which we will be predicting.



	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a
5	e	x	y	y	t	a	f	c	b	n	...	s	w	w	p	w	o	p	k	n
6	e	b	s	w	t	a	f	c	b	g	...	s	w	w	p	w	o	p	k	n
7	e	b	y	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	s
8	p	x	y	w	t	p	f	c	n	p	...	s	w	w	p	w	o	p	k	v
9	e	b	s	y	t	a	f	c	b	g	...	s	w	w	p	w	o	p	k	s

Fig 1: Illustration of data in dataset

To check if there are some null values present, we performed a simple test which gave us negative result. This is a good sign and tells us that our dataset is well written and almost ready for our classification.

Target variable class values are present in almost balanced ratio 4208:3916. This is also important factor because we can continue with it without using some over/undersampling methods.

Features in dataset don't have fixed number of unique values. For example, feature "bruises" only has 2 unique variables, but "cap-color" has 10. That means that some feature could possibly tell us more about mushroom edibility. In the histogram below, I showed that odor can be somehow connected to the edibility; this could be true for more features, but of course, it's not for sure.

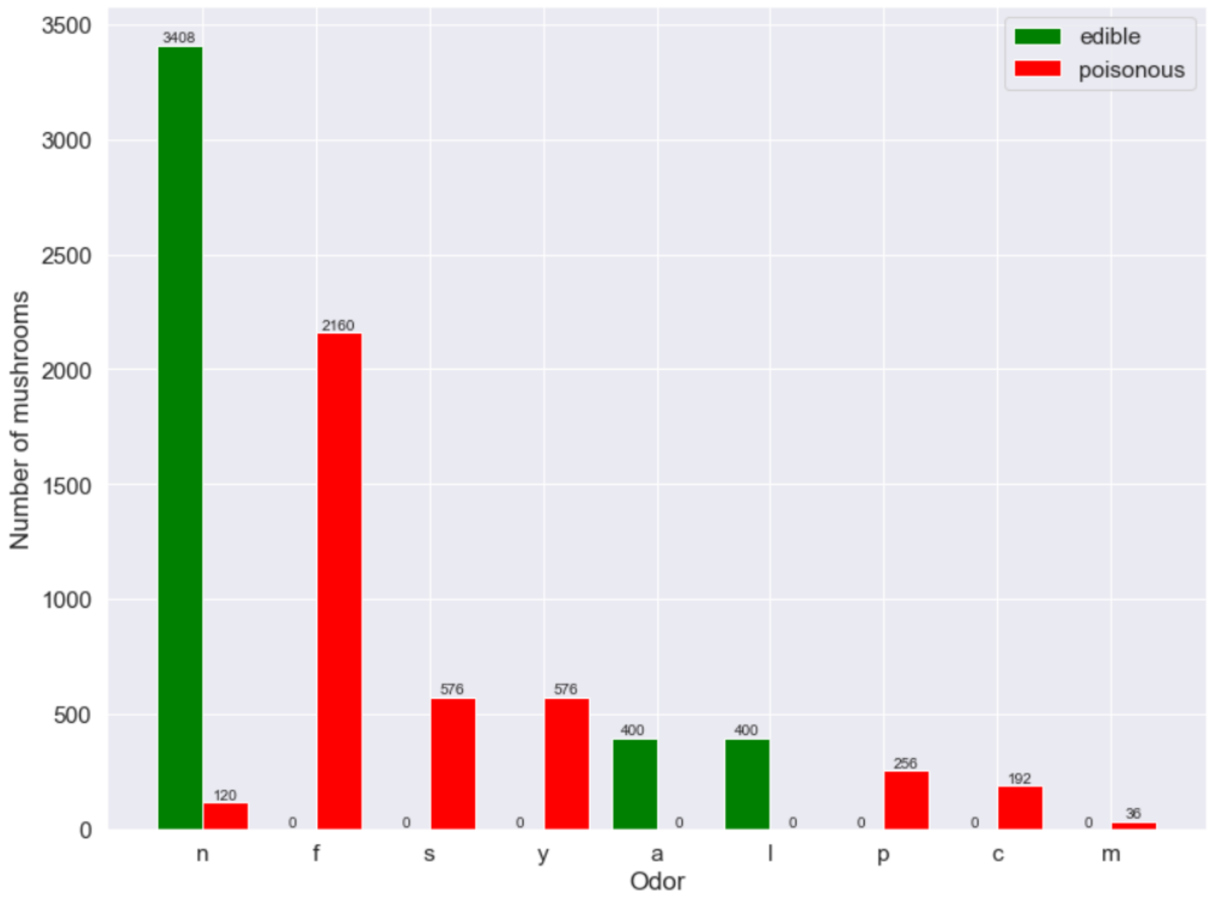


Fig 2: Histogram for features "odor" and "class" – edibility

All this data analyzing proved us that our dataset contains correct data, has many interesting features and can be used for a classification process – but firstly, we need to prepare it.

3. Data preprocessing

Before we can proceed with classification, we need to change structure of our dataset.

Firstly, we need to change data from characters to integers – to this, we used LabelEncoder for easy manipulation.

If we look closely at our data, we can see that features are representing something called Nominal categorical variables. These are the variables which cannot be somehow sorted. For example, we cannot say if this color is greater than that color. If we would use our variables without somehow solving this problem, our estimator would think that these features are somehow ordered and that's not something we want. To solve this, we use one-hot encoding (with dummy variables), which creates n columns based on n unique values.

Now we could perform feature scaling and lastly PCA dimension reduction with 2 components. With this, we can easily visualize all decision boundaries of our models.

4. Models and their parameters

With our dataset ready we can perform classification. I decided to use 3 different algorithms with first two of them being Support Vector Machines and Random Forrest, as proposed. Unfortunately, in my project proposal I made a silly mistake and presented Convolutional Neural Network as third algorithm to be used, which is, of course, not logical for our classification. It was pointed out to me by teacher (thanks for that!) and I recognized my mistake. To correct myself, I decided to use another algorithm: Logistic regression.

To make my project more interesting, I will use every algorithm 2 times. First time, our estimator will be presented with no custom parameters apart from state (which will be always set to 42 – answer to everything) and parameters which are needed to be set to silence warnings (e.g gamma in SVM). That means every parameter will be set to some automatic value. Every algorithm will try to learn from 75% of the dataset (training data) and then will be tested on the remaining 25% (testing data).

Second time, before training begins, I will try to find best parameters for each estimator. That means I declare some values for important parameters for each model and let the program decide which values brings us best results and if it even helps somehow. For deciding (that mean finding parameters) I use sklearn feature GridSearchCV. This was a big change of plans for me, because initially I wanted to use RandomSearchCV, slightly different, but faster variation, but after getting into some problems (see Additional info for more), I decided to use the first mentioned.

After training and testing, I extract all important information about results and visualize decision boundary and confusion matrix of each model.

5. Classification, visualization and results

Firstly, I quickly describe final parameters of each model, then proceed to results and at the end I show resulting visualizations.

a) Support Vector Machines (SVM)

- simple model
  In the simple model of SVM, I needed to declare gamma to silence warning. I set it to ‘auto’. Other than that, the model was ready to be trained and tested. Details about this model:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=42, shrinking=True, tol=0.00,
  verbose=False)
```

- model with custom parameters
  I decided to look for best values in parameters C (regularization parameters), gamma (coefficient for some types of kernel) and kernel. Resulting model

(with best parameters from values I offered to model, see Additional info for more) is:

```
SVC(C=1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf',
max_iter=    -1, probability=False, random_state=42, shrinking=True,
tol=0.001, verbose=False)
```

b) Random Forrest
- simple model
  Once again, I was forced to declare one parameter, and that is n_estimators, which I set to 10. Other details about this model:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10,
            n_jobs=None, oob_score=False, random_state=42, verbose=0,
            warm_start=False)
```

- model with custom parameters
  In Random Forest model, I decided too look for best values in parameters n_estimators (number of trees in forest), criterion (function measuring quality of split), max_features (number of features when looking for split) and min_samples_leaf (minimum number of samples to be leaf). After finding these parameters, model looks like this:

```
RandomForestClassifier(bootstrap=True, class_weight=None,
            criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=60,
            n_jobs=None, oob_score=False, random_state=42, verbose=0,
            warm_start=False)
```

c) Logistic Regression
- simple model
  Without setting parameter solver, I wouldn't be able to silence warning, so I set it to ‚liblinear'. Simple model parameters are:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='warn',
n_jobs=None, penalty='l2', random_state=42, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
```

- model with custom parameters
  In Logistic Regression model, I tried to find best values among parameters C (regularization parameter), penalty (norm of penalization) and solver (algorithm used for optimalization problem). With best values of these parameters, resulting model is:

  LogisticRegression(C=1, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=42, solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

Now for the results of every model, I decided to write here just some of the important values. For all calculated values, see file with code or section Additional info. The most important values, accuracies, are also different in colors (green – best, red – worst).

Here are the results:

Process	Training			Testing
Model\Category	Accuracy	Average Accuracy	Standard Deviation	Accuracy
Simple SVM	92.93%	92.88%	0.0089	92.71%
Custom SVM	93.27%	93.25%	0.0084	92.91%
Simple RF	98.87%	92.37%	0.0111	92.61%
Custom RF	94.76%	93.29%	0.0090	93.11%
Simple LR	90.58%	90.58%	0.0095	90.35%
Custom LR	90.58%	90.56%	0.0097	90.35%

Tab 1: Results of classification by 6 models

From results we can see that the best simple and custom model was Random Forest. The difference between Random Forest and Support Vector machines is not high though – the only model among the three which didn't get at least 92% in any accuracy is Logistic Regression. This could have been probably expected though.

One of the very interesting things to notice is really big training accuracy of simple Random Forest. Unfortunately, that doesn't tell us the whole story – testing accuracy showed us true colors. Nevertheless, custom Random Forest made up for this and it's testing accuracy improved simple testing accuracy by 0.5 %.

Both Support Vector Machines and Random Forest showed us (at least in my opinion) very good testing accuracy. 93% is in my opinion great number. Although Logistic Regression fell behind, it's 90% is still really impressive and the adjective "worst" is, in fact, "still good".

Results also show us importance of good parameters – it seems like finding some of them really helped. For example, already mentioned testing accuracies of simple and custom Random Forrest differ in half percent, which can seem low, but if we imagine this in a real world example, that could save our lives! (more so if precision of detecting poisonous mushrooms is higher – see Additional info part!)

All interesting visualization is present in the file with code of this project. Here I will show three graphs – all are visualizing decision boundary of testing of all 3 various models with custom parameters:

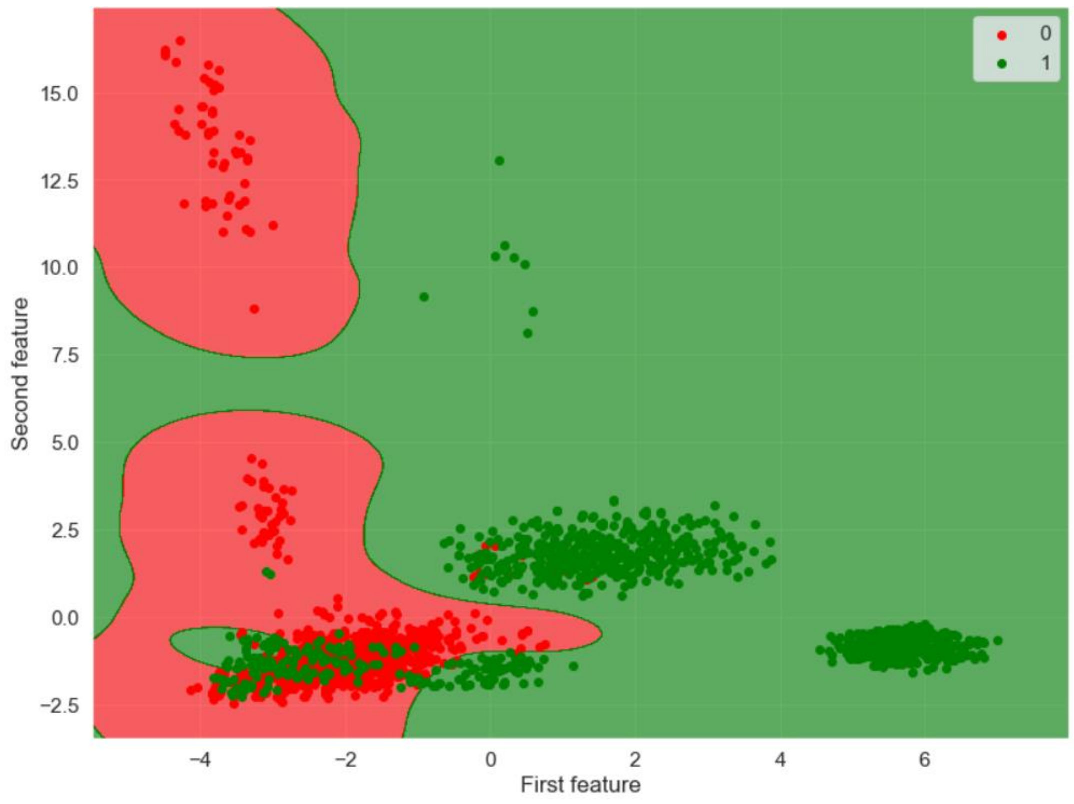


Fig 3: Testing decision boundary visualization of custom SVM model

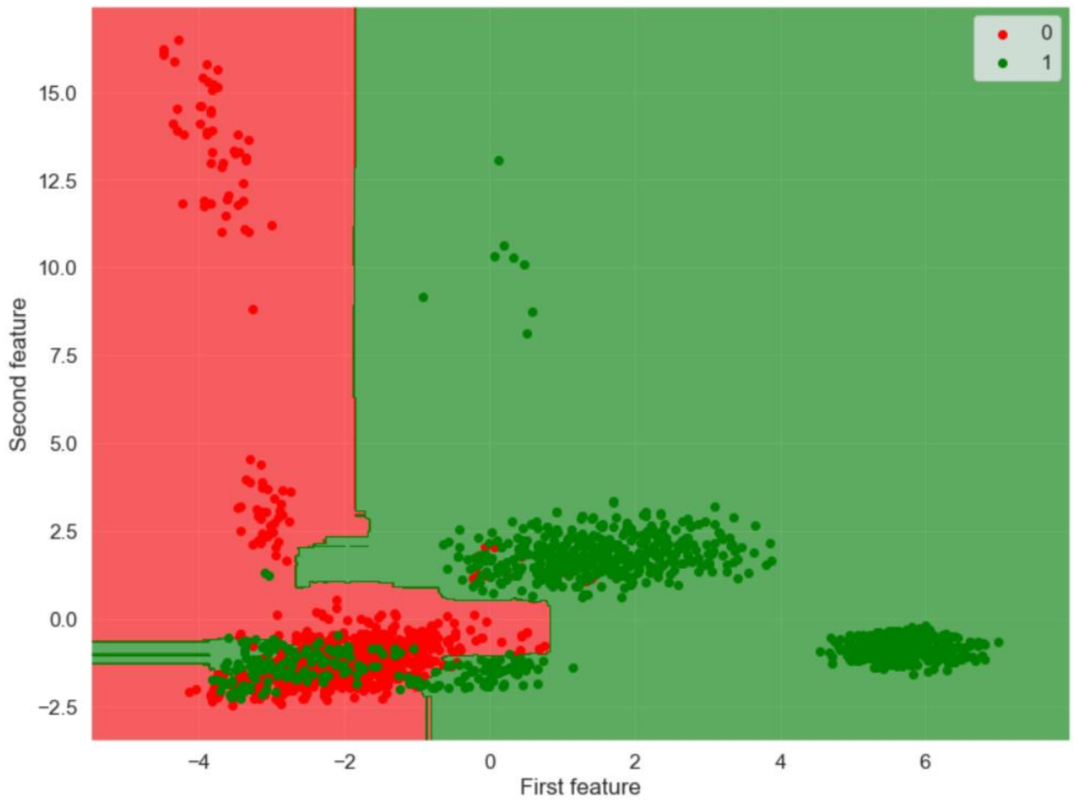


Fig 4: Testing decision boundary visualization of custom RF model

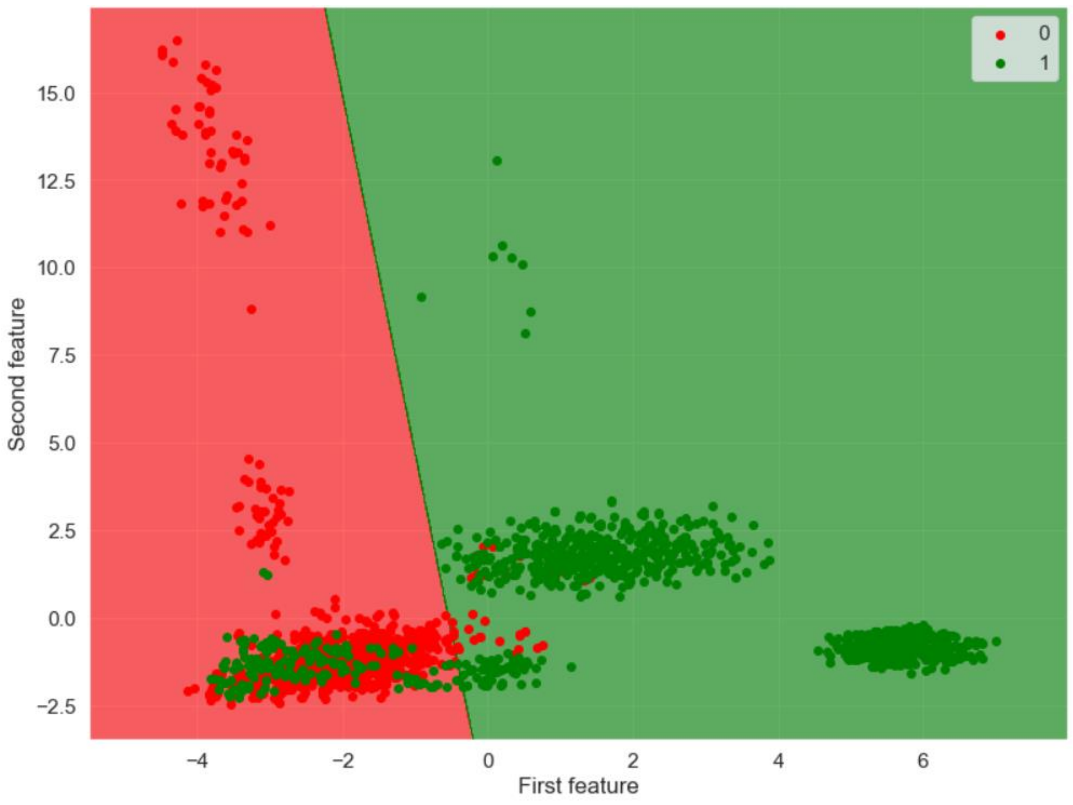


Fig 5: Testing decision boundary visualization of custom LR model

6. Conclusion

We loaded a dataset and analyzed it. Analysis confirmed that dataset is good for applying machine learning technics on, we had to edit it a little bit though. With using of one-hot encoding, feature scaling a pca dimension reduction we got workable data. We applied on this data 3 different classification algorithms – Support Vector Machines, Random Forrest and Logistic Regression, each two times with different parameters to get best results. Random Forrest proved to be the best option with Support Vector Machines not far behind in terms of accuracy percentage. Logistic Regression fell behind a little bit, but still got respectable accuracy of 90+ %.

Visualization helped us imagine how each model “thought” and what’s difference between each of them. Results are really good and high percentage not only proved correctness of dataset and classification algorithms, but also whole composition and process of making this project.

7. Additional info

This part is not only about our project anymore, but rather covers some small details about implementation, sources and other interesting things.

Firstly, it's important to remind that most of the things written here should be correct, but can happen some mistake was made. Most of the information comes from lectures and tutorials of Machine learning on Faculty mathematics, physics and informatics on Comenius university, official documentations of used modules, mostly sklearn and different online courses and guides. Some specifics links on the website are present below, but the whole documentation of sklearn and lecture notes of Machine learning are good sources for additional information on anything used and showed in this project.

In the source code, there are several superscripts (numbers) representing some problem method, which should be a little bit more described. We can look at them now:

1. I did not come up with this code alone, but it works as intented. Source is here: https://stackoverflow.com/a/41646557
2. Most of these imports were written one by one when needed, some of them were change, some deleted completely. It's possible to do many things without these specific libraries – but it made writing this project a lot easier. Most of imported libraries and modules come from sklearn: https://scikit-learn.org/stable/
3. Some parts of code were inspired by various sources mentioned above, but most of it is work of this project. That means it's not that much tested and could possibly be flawed in some devices or different version of programs/modules. Also, some of the code is specifically designed for this dataset, especially functions for plotting first 2 graphs.
4. In finding best parameters with the help of GridSearchCV (or RandomizedSearchCV) cv parameter must be specified to silence warnings.
5. When using pandas.Dataframe and seaborn.heatmap it is possible to encounter problem where values are not centered (they are on some corner or barely visible part of the square). After trying to find how to prevent this, it was discovered that this problem appears in some new version of matplotlib. To fix this issue, matplotlib must be downgraded to the version 3.1.0. or older. This can also change design of some graphs, but just slightly.
6. One-hot enconding is usually used when dealing with nominal categories variables. One source which helped to understand this problem and the reason of solving it way used in project: https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d

7. Once again, thanks for correction after project proposal. Logistic regression fits here perfectly and at least it was one new algorithm, which wasn't tried on tutorials. All information about implementation comes from official documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
8. Parameters which were taken into consideration when trying to make best model possible were determined by reading official documentation for each model. Values present for each of these parameters were mainly based on official documentation and some real implementations of different classification models (for different problems). One must keep in mind that not all combinations make sense: for example in Logistic Regression solvers 'newton-cg', 'sag' and 'lbfgs' support only L2 regularization, so it would make no sense trying to find model with one of these solvers and L1 regularization. Another example is gamma in Support Vector Machines – it's coefficient for 'rbf', 'poly' and 'sigmoid' kernels, so no need to declare some values when also 'linear' is present. In this case to silence warnings, it had to be declared as auto when no values was assigned though. Everything important about what is possible and why can be it good or bad is in documentation for our 3 estimators:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
9. In the whole process of making this project, RandomizedSearchCV was preferred and used for finding best hyperparameters. The biggest reason why is time duration – RandomizedSearchCV took about 30 seconds, on the other hand GridSearchCV takes minutes to finish, especially on Random Forrest. The reason why is GridSearchCV used in the final version is simple – it works. When finding parameters for model, searching tool is presented with either dictionary (when all values of all parameters are free to mix) or list of dictionaries (when specific values have specific requirements; see 8 for more). Unfortunately, even though official documentation of RandomizedSearchCV supports list of dictonaries as a parameter, in fact RandomizedSearchCV works only with dictionary as parameter. This issue was known and addressed to sklearn, which supposedly solved this issue, but we couldn't make it work in our project. That's the reason why is slower GridSearchCV used instead. Some interesting links addressing this issue are provided below:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
https://github.com/scikit-learn/scikit-learn/issues/12728

In case anything is missing or not clear, feel free to ask!

Last part it about results. It’s important to note that definitions of many terms (e.g. precision, recall rate, confusion matrix etc.) are not present in this document – it is assumed that these definitions are well-known for any reader. There are also many articles on the internet where these concepts are well-documented and explained.

When finding best values for parameters of models, these were values to be chosen from:

a) SVM
   - C: 0.01, 1, 10, 100, 500
   - gamma: auto/1, 0.1, 0.01, 0.001, 0.0001
   - kernel: linear, rbf/rbf

b) RF
   - max_features: ‘auto’, ‘sqrt’, ‘log2’
   - criterion: ‘gini’, ‘entropy’
   - n_estimators: 10, 30, 50, 60, 70, 80, 90, 100
   - min_samples_leaf: 1, 5, 10, 20, 40, 60, 90, 100

c) LR
   - C: 0.001, 0.01, 0.1, 1, 10, 100, 500
   - penalty: L1/L2
   - solver: ‘liblinear’, ‘saga’/’liblinear’, ‘saga’, ‘newton-cg’, ‘sag’, ‘lbfgs’

All results of training and testing models used in this project are printed in almost original format below, most of the data were produced by function from sklearn.metrics library:

a) SVM simple model:

```
Training results:
Accuracy: 0.9292630887904152
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.98      0.93      3168
           1       0.97      0.88      0.92      2925
    accuracy                           0.93      6093
   macro avg       0.93      0.93      0.93      6093
weighted avg       0.93      0.93      0.93      6093
Confusion Matrix:
[[3097   71]
 [ 360 2565]]
Average Accuracy: 0.9287721671334923
Standard Deviation: 0.008890483871605827
------------------------------------------------------------
```

```
Testing results:
Accuracy: 0.9271294928606598
Classification Report:
              precision    recall  f1-score   support
           0       0.89      0.98      0.93      1040
           1       0.97      0.88      0.92       991
    accuracy                           0.93      2031
   macro avg       0.93      0.93      0.93      2031
weighted avg       0.93      0.93      0.93      2031
Confusion Matrix:
[[1015   25]
 [ 123  868]]
```

b) SVM custom model (C = 1, gamma = 1, kernel = 'rbf'):

```
Training results:
Accuracy: 0.9327096668307895
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      3168
           1       0.97      0.89      0.93      2925
    accuracy                           0.93      6093
   macro avg       0.94      0.93      0.93      6093
weighted avg       0.94      0.93      0.93      6093
Confusion Matrix:
[[3078   90]
 [ 320 2605]]
Average Accuracy: 0.9325461583580206
Standard Deviation: 0.008440418936419098
-----------------------------------------------------------

Testing results:
Accuracy: 0.9290989660265879
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      1040
           1       0.96      0.89      0.92       991
    accuracy                           0.93      2031
   macro avg       0.93      0.93      0.93      2031
weighted avg       0.93      0.93      0.93      2031
Confusion Matrix:
[[1006   34]
 [ 110  881]]
```

c) Random Forest simple model:

```
Training results:
Accuracy: 0.9886755292959133
Classification Report:
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3168
           1       1.00      0.98      0.99      2925
    accuracy                           0.99      6093
   macro avg       0.99      0.99      0.99      6093
weighted avg       0.99      0.99      0.99      6093
Confusion Matrix:
[[3161    7]
 [  62 2863]]
Average Accuracy: 0.9236840006672985
Standard Deviation: 0.011124047729564948
-------------------------------------------------------

Testing results:
Accuracy: 0.9261447562776958
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      1040
           1       0.95      0.89      0.92       991
    accuracy                           0.93      2031
   macro avg       0.93      0.93      0.93      2031
weighted avg       0.93      0.93      0.93      2031
Confusion Matrix:
[[995  45]
 [105 886]]
```

d) Random Forest custom model (criterion = ‘entropy‘, max_features = ‘auto‘, min_samples_leaf = 5, n_estimators = 60)

```
Training results:
Accuracy: 0.9476448383390776
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.98      0.95      3168
           1       0.97      0.92      0.94      2925
    accuracy                           0.95      6093
   macro avg       0.95      0.95      0.95      6093
weighted avg       0.95      0.95      0.95      6093
```

```
Confusion Matrix:
[[3092   76]
 [ 243 2682]]
Average Accuracy: 0.9328769935721157
Standard Deviation: 0.009036430542338788
------------------------------------------------------------
Testing results:
Accuracy: 0.931068439192516
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.93      1040
           1       0.96      0.90      0.93       991
    accuracy                           0.93      2031
   macro avg       0.93      0.93      0.93      2031
weighted avg       0.93      0.93      0.93      2031
Confusion Matrix:
[[1002   38]
 [ 102  889]]
```

e) Logistic Regression simple model

```
Training results:
Accuracy: 0.9057935335631052
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.97      0.91      3168
           1       0.96      0.84      0.90      2925
    accuracy                           0.91      6093
   macro avg       0.91      0.90      0.90      6093
weighted avg       0.91      0.91      0.91      6093
Confusion Matrix:
[[3066  102]
 [ 472 2453]]
Average Accuracy: 0.9057957473177007
Standard Deviation: 0.00946748882265365
------------------------------------------------------------
Testing results:
Accuracy: 0.9034958148695224
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      1040
           1       0.96      0.84      0.89       991
    accuracy                           0.90      2031
   macro avg       0.91      0.90      0.90      2031
weighted avg       0.91      0.90      0.90      2031
Confusion Matrix:
[[1003   37]
 [ 159  832]]
```

f) Logistic Regression custom model (C = 1, penalty = ‘L1‘, solver = ‘liblinear‘)

```
Training results:
Accuracy: 0.9057935335631052
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.97      0.91      3168
           1       0.96      0.84      0.90      2925
    accuracy                           0.91      6093
   macro avg       0.91      0.90      0.90      6093
weighted avg       0.91      0.91      0.91      6093
Confusion Matrix:
[[3066  102]
 [ 472 2453]]
Average Accuracy: 0.90563127363349
Standard Deviation: 0.009670295068378272
------------------------------------------------------------
Testing results:
Accuracy: 0.9034958148695224
Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.96      0.91      1040
           1       0.96      0.84      0.89       991
    accuracy                           0.90      2031
   macro avg       0.91      0.90      0.90      2031
weighted avg       0.91      0.90      0.90      2031
Confusion Matrix:
[[1003   37]
 [ 159  832]]
```