ITMO UNIVERSITY

Saint Petersburg, Russia

Using the FEDOT framework functionality for the robotics task

Yachmenkov Mikhail

tutorial

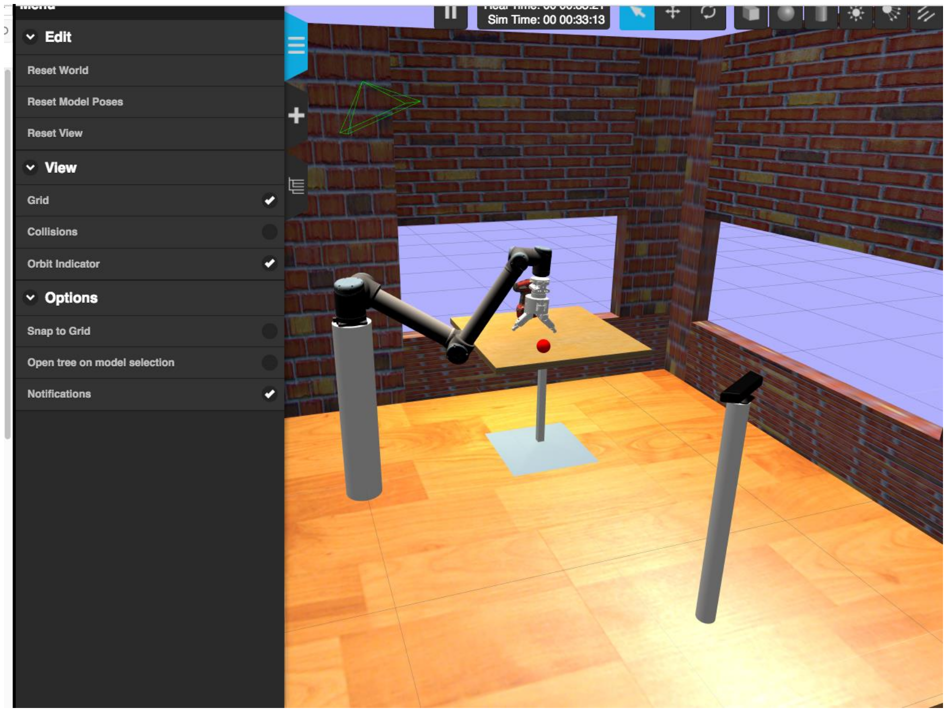
Case study - grasp stability prediction

УНИВЕРСИТЕТ ИТМО

The case which was dedicated to the problem of manipulator grasp stability binary classification at the Kaggle platform is used (https://www.kaggle.com/ugocupcic/grasping-dataset)

The main goal was to predict grasp stability based on several features like positions, angular velocities, efforts of each kinematic joint. The proposed manipulator consisted of one hand with three fingers with three joints per the last one.

Total amount of joints was equal to 9 and therefore total amount of features was equal to 27.



https://github.com/shadow-robot/smart_grasping_sandbox

Modify from_csv metod


```diff
     @staticmethod
-    def from_csv(file_path, delimiter=',',
+    def from_csv(file_path, headers=[], delimiter=',',
                  task: Task = Task(TaskTypesEnum.classification),
                  data_type: DataTypesEnum = DataTypesEnum.table,
-                 with_target=True):
+                 with_target=True, target_header=''):
         data_frame = pd.read_csv(file_path, sep=delimiter)
+        data_frame.columns = [i.strip(" ") for i in data_frame.keys()]
+        data_frame.drop(headers, axis='columns', inplace=True)
         data_frame = _convert_dtypes(data_frame=data_frame)
         data_array = np.array(data_frame).T
         idx = data_array[0]
         if with_target:
-            features = data_array[1:-1].T
-            target = data_array[-1].astype(np.float)
+            if target_header:
+                target = np.array(data_frame[target_header]).astype(np.float)
+                pos = list(data_frame.keys()).index(target_header)
+                features = np.delete(data_array.T, pos, axis=1)
```

Modify of from_csv metod

УНИВЕРСИТЕТ ИТМО

```diff
+            else:
+                target = data_array[-1].astype(np.float)
+                features = data_array[1:-1].T
         else:
             features = data_array[1:].T
             target = None
-        return InputData(idx=idx, features=features, target=target, task=task, data_type=data_type)
+        return [InputData(idx=idx, features=features, target=target, task=task, data_type=data_type), data_frame]
```

Modify evolution parameters

$P_{crossover}$	$P_{mutation}$	d_{max}	a_{max}	$n_{generation}$	$s_{population}$
0.8	0.8	3	3	20	20

ITMOre than a UNIVERSITY

Add accuracy metric

УНИВЕРСИТЕТ ИТМО

```
class AccuracyScore(Chain):
    @staticmethod
    @from_maximised_metric
    def get_value(chain: Chain, reference_data: InputData) -> float:
        try:
            # validate(chain)
            results = chain.predict(reference_data)
            y_pred = [round(predict) for predict in results.predict]
            score = round(accuracy_score(y_true=reference_data.target.round().astype(int).tolist(),
                                         y_pred=y_pred), 3)
        except Exception as ex:
            print(ex)
            score = 0.5

        return score
```

ITsMOre than a UNIVERSITY

Performance model estimation


```
def create_performance_model(dataset: InputData, chain: Chain, path_to_save: str,
                             time_limit: int, percent: float, top_percent: float,
                             percent_step: float, feature_top=None, feature_step: int = 1,
                             path_to_save_figure: str = None, n: int = 1) -> pd.DataFrame:
    initial_time = datetime.datetime.now()
    current_time = 0
    arr = []
    massive = []
    features_count = dataset.features.shape[1]
    if feature_top is None:
        feature_top = features_count
    if feature_top > features_count:
        raise ValueError('Invalid value of param feature_top')
    dataset_original = deepcopy(dataset)
    initial_percent = percent
    for i in np.arange(1, feature_top+1, feature_step):
        dataset.features = dataset_original.features[:, :i]
        while current_time <= time_limit and percent <= top_percent:
            # decreasing number of dataset lines
            data, _ = train_test_data_setup(dataset, split_ratio=percent)
            num_lines = data.target.shape[0]
...
```

```
            # n-times fitting in a fixed (x, y) point of performance model
            time_local = []
            for j in range(n):
                # calculate parameters optimization time for a given chain
                start_time = datetime.datetime.now()
                chain.fit(input_data=data, verbose=True)
                time = datetime.datetime.now() - start_time
                arr.append([time.total_seconds(), num_lines, i, j])
                time_local.append(time.total_seconds())
            time_mean = np.array(time_local).mean()
            massive.append([time_mean, num_lines, i])

            percent += percent_step
            current_time = round((datetime.datetime.now() -
initial_time).seconds / 60)
        percent = initial_percent
```

Estimate performance models (1/2)

УНИВЕРСИТЕТ ИТМО

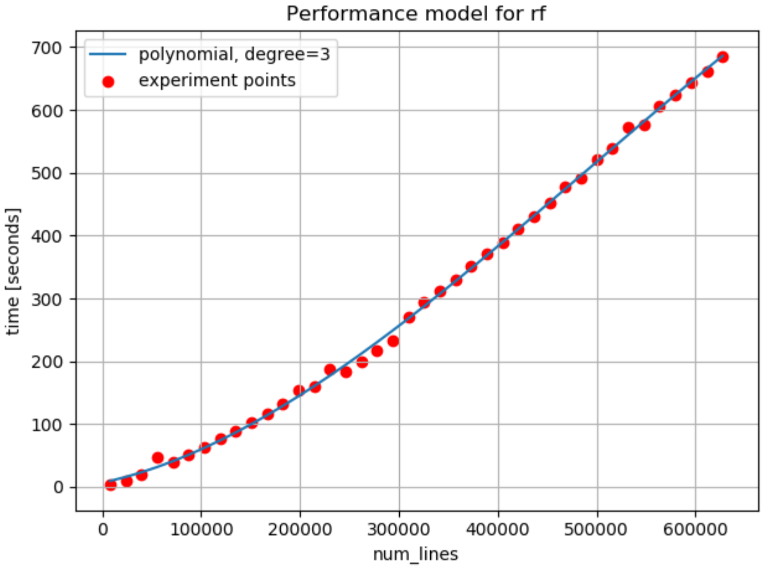


Figure 1 – y=a0*x^3+a1*x^2+a2*x+a3,
Koeff=[-1.39e-15, 2.06e-09, 3.39e-04, 6.752]

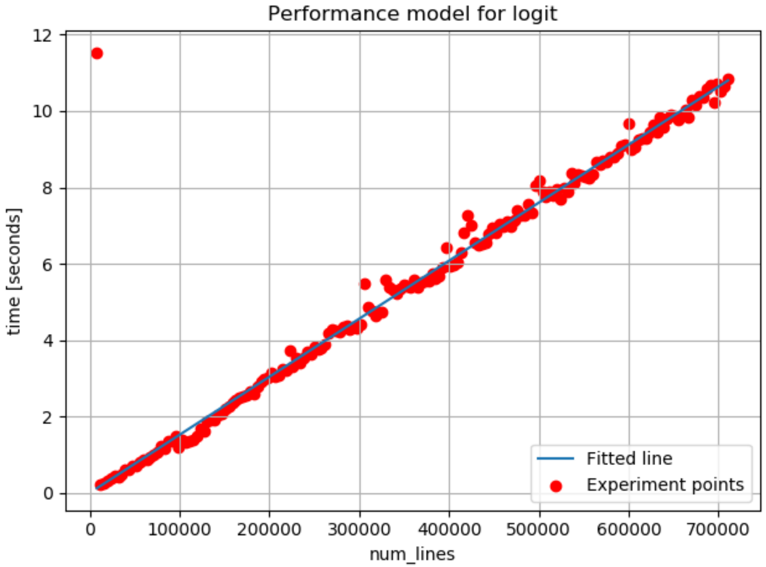


Figure 2 – y=k*x, k=1.5e-05

ITsMOre than a UNIVERSITY

Estimate performance models (2/2)

УНИВЕРСИТЕТ ИТМО

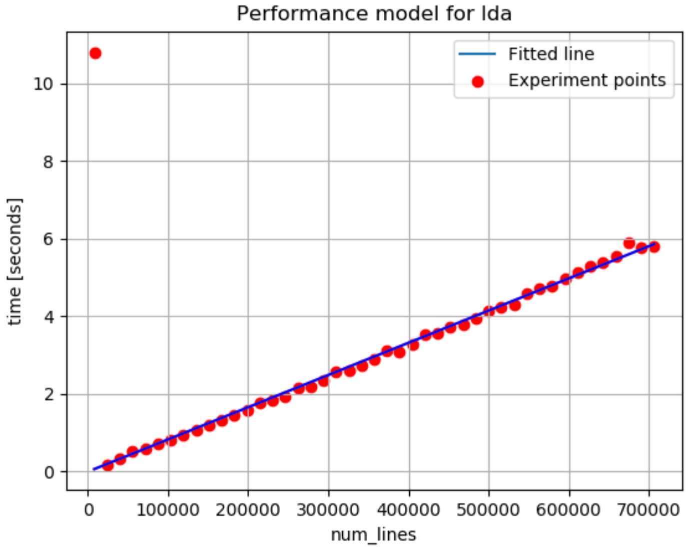


Рисунок 3 – y=k*x, k=8.2e-06

ITMOre than a UNIVERSITY

Composite model identification (1/2)

```
if part_of_dataset_to_compose >= 0.8:
        raise ValueError("The argument part_of_dataset_to_compose has to be less than 0.8")
    dataset_train_fit = None
    task = Task(TaskTypesEnum.classification)
    dataset = InputData.from_csv(dataset_path, headers=['measurement_number'], task=task, target_header='robustness')

    # this is a sensible grasp threshold for stability
    good_grasp_threshold = 100

    # divide the grasp quality on stable or unstable grasps
    dataset.target = np.array([int(i > good_grasp_threshold) for i in dataset.target])

    # split dataset to train and test sets
    if full_dataset:
        dataset_to_compose, dataset_to_validate = train_test_data_setup(dataset)
        part_of_dataset_to_compose = 0.8
    else:
        # decreasing dataset size to accelerate composing
        dataset_train_fit, dataset_to_validate = train_test_data_setup(dataset)
        dataset_to_compose, _ = train_test_data_setup(dataset_train_fit, split_ratio=part_of_dataset_to_compose/0.8)
```

Composite model identification (2/2)

УНИВЕРСИТЕТ ИТМО

```
# the search of the models provided by the framework that can be used as nodes in a chain for the selected task
    available_model_types, _ = ModelTypesRepository().suitable_model(task_type=task.task_type)

    # the choice of the metric for the chain quality assessment during composition
    metric_function = MetricsRepository().metric_by_id(ClassificationMetricsEnum.ROCAUC_penalty)

    # the choice and initialisation of the GP search
    composer_requirements = GPComposerRequirements(
        primary=available_model_types,
        secondary=available_model_types, max_arity=2,
        max_depth=3, pop_size=20, num_of_generations=20,
        crossover_prob=0.8, mutation_prob=0.8,
        max_lead_time=max_lead_time, add_single_model_chains=False)

    # Create GP-based composer
    composer = GPComposer()
    print(f'Dataset size to compose: {part_of_dataset_to_compose * 100}%')

    # the optimal chain generation by composition - the most time-consuming task
    chain_evo_composed = composer.compose_chain(data=dataset_to_compose,
                                                initial_chain=None,
                                                composer_requirements=composer_requirements,
                                                metrics=metric_function,
                                                is_visualise=is_visualise
```

Analyse the solutions

УНИВЕРСИТЕТ ИТМО

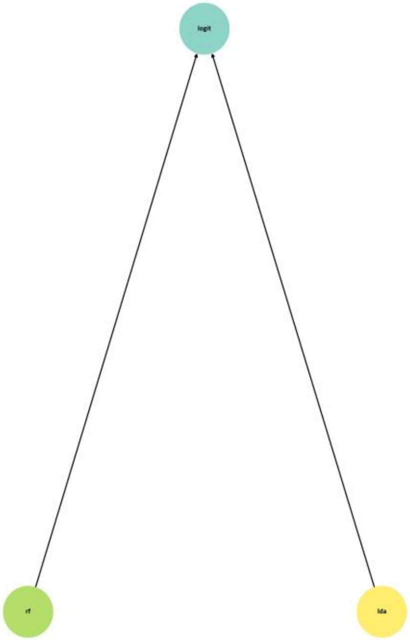


Figure 4 – logit_lda_rf

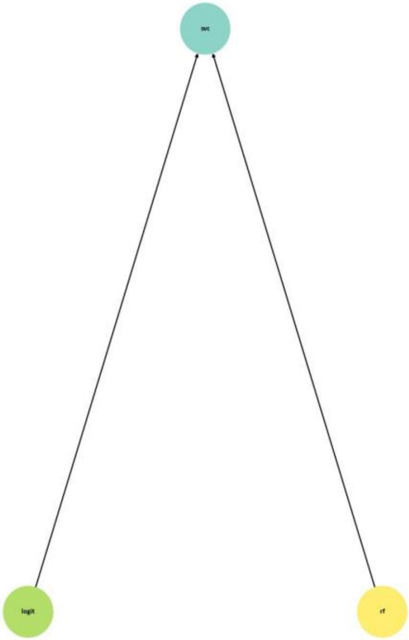
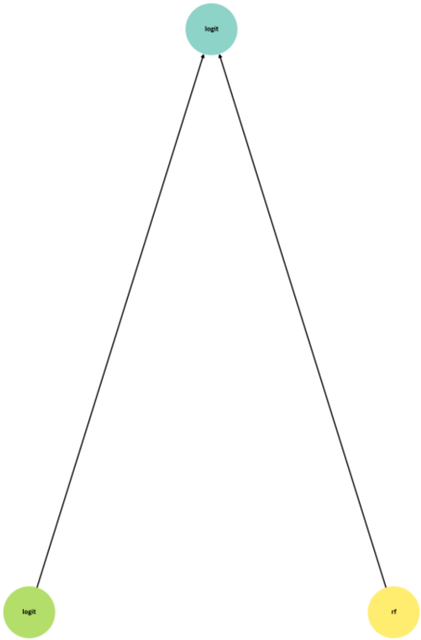


Figure 5 – svc_logit_rf



Рису Figure нок 6 – logit_logit_rf

Figure 7 – rf

ITsMOre than a UNIVERSITY

Analyze the convergence

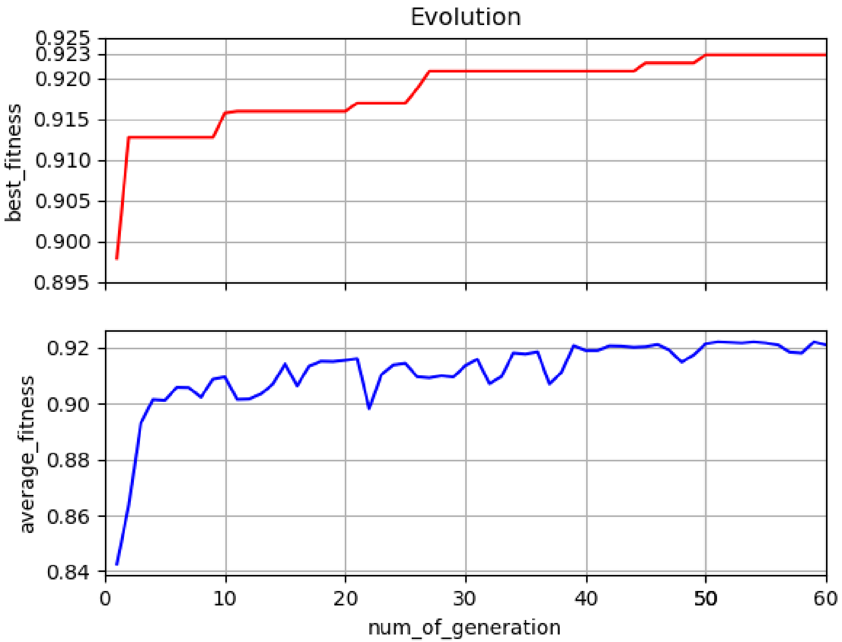


Figure 8 – logit_logit_rf

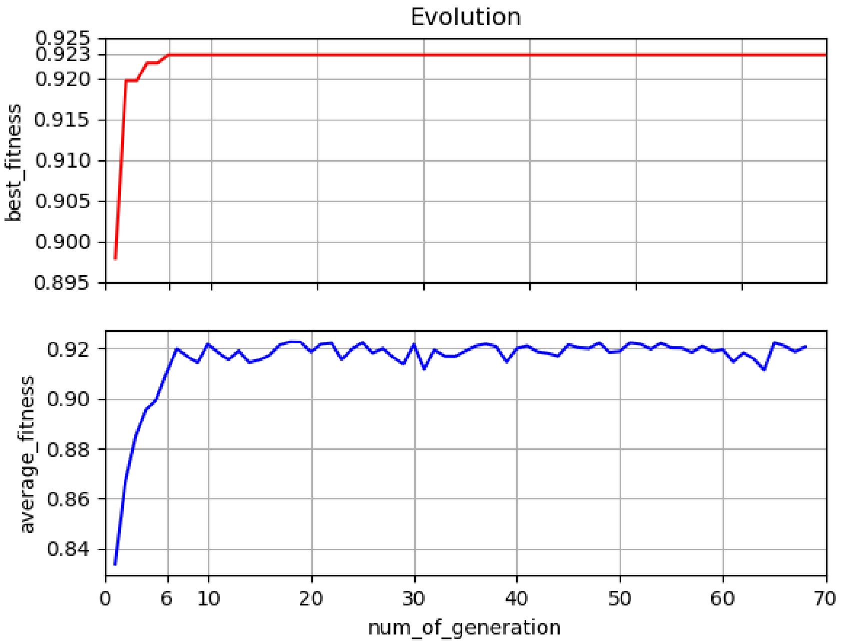


Figure 9 – logit_lda_rf

Analyze metric

Table 1 – Modelling quality metrics

	Log_log_rf	Log_rf_lda	Svc_log_rf	rf	NN_baseline	LightGBM
Accuracy	0.9700	0.9700	0.9700	0.9670	0.7867	0.9586
Roc_auc	0.9960	0.9960	0.9960	0.9960	0.7973	0.9605
f1	0.9670	0.9669	0.9663	0.9562	0.7934	0.9553
Precision	0.9521	0.9521	0.9496	0.9403	0.7053	0.9323
Recall	0.9835	0.9834	0.9842	0.9862	0.9067	0.9795

Refine the performance models (1/3)

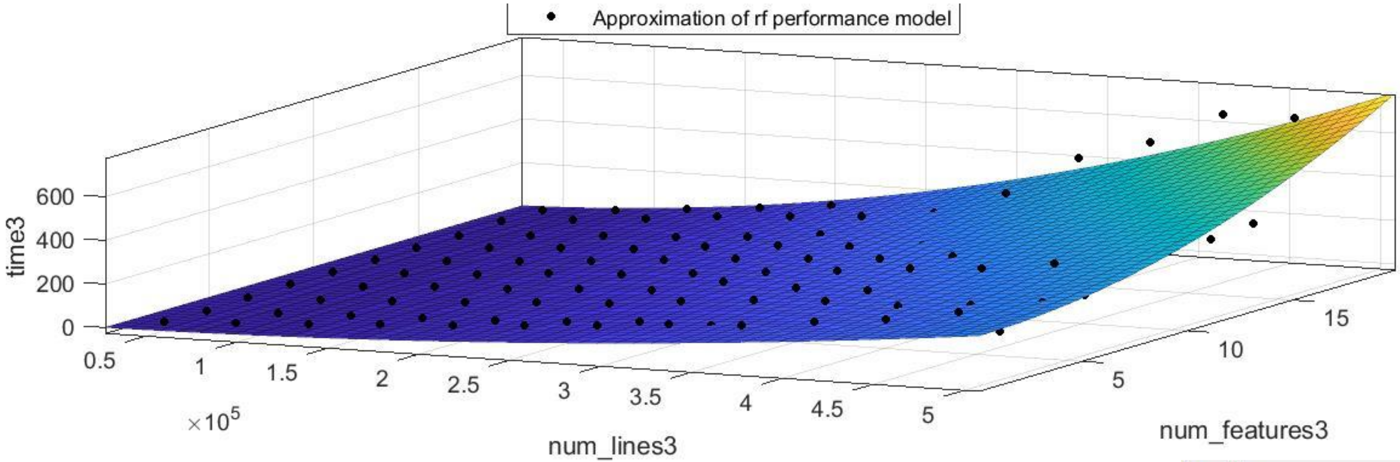


Figure 10 – Performance model for the random forest

Results

General model:

f(x,y) = x^2/a^2+x^2*y^2/b^2

Coefficients (with 95% confidence bounds):

a = 3.365e+04 (3.106e+04, 3.625e+04)

b = 4.282e+05 (3.927e+05, 4.637e+05)

Goodness of fit:

SSE: 3.66e+05

R-square: 0.7959

Adjusted R-square: 0.7941

RMSE: 56.66

Refine the performance models (2/3)

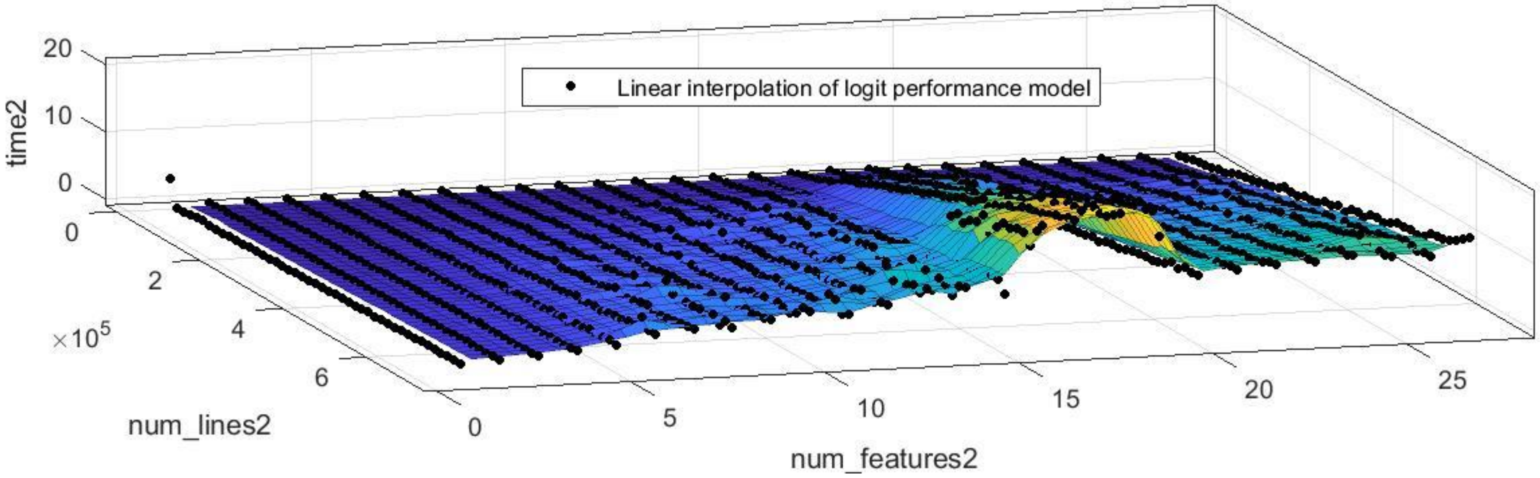


Figure 11 – Performance model for the logit

Refine the performance models (3/3)

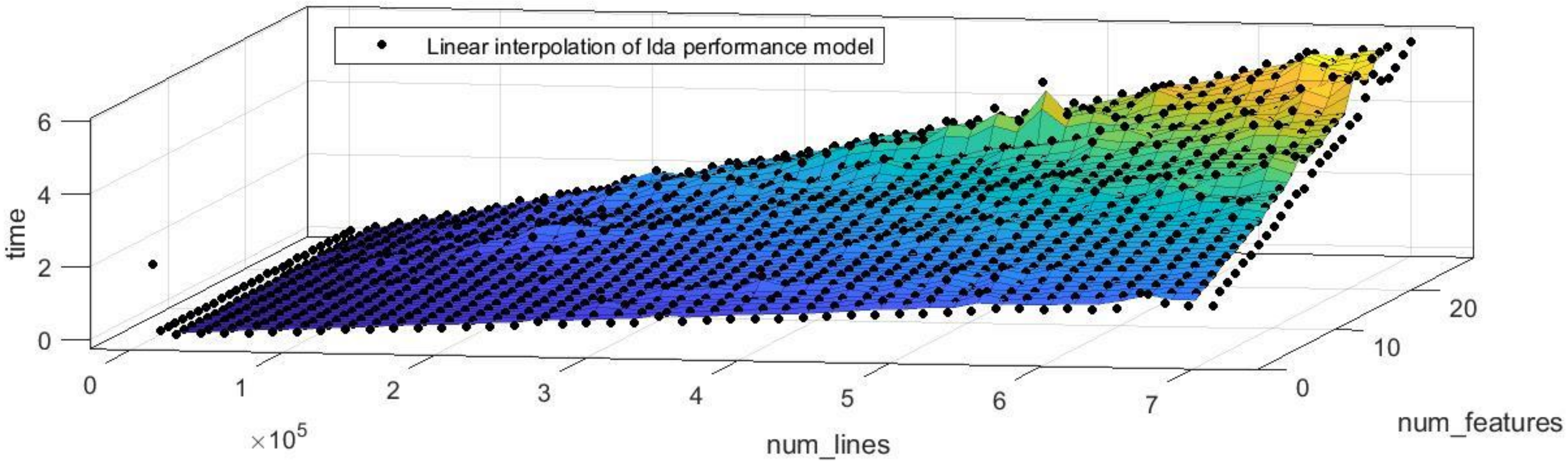
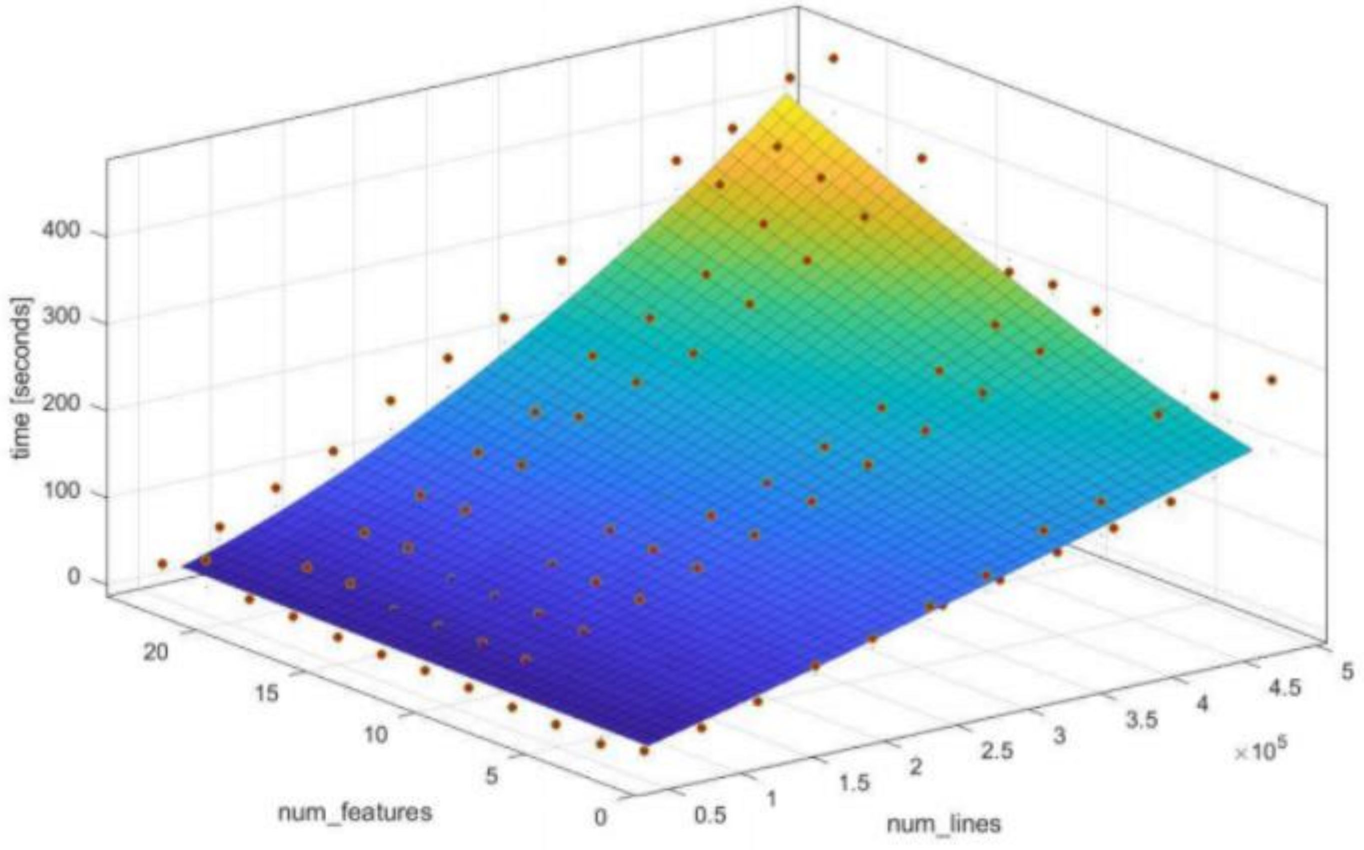


Figure 12 – Performance model for the lda

Build model for the composite chain



Conclusion

УНИВЕРСИТЕТ ИТМО

- Firstly, the optimal in terms of a grasp stability prediction chains with enhanced performance with respect to existing methods have been proposed. It should be mentioned that just a single rf model give an excellent results but metrics values can be improved by including genetic algorithm to compose not single model chain.

- Secondly, it was proposed chain performance model extrapolation procedure with a good enough validation metrics values. It can be used to detect problems with chain fitting algorithms like time redundancy of a chain fit process or to intelligently manage of automate models generation.

- Further, it can be conducted some experiments to mix obtained best chains into a single chain to potentially improve quality. Besides, the dataset can be extended by new simulations with new grasping objects of different shapes.