ITMO UNIVERSITY

Saint Petersburg, Russia

Using the FEDOT framework functionality for the economics task

Timur Kutuev

tutorial

Analysis of fraudulent operations with bank cards

ITMO UNIVERSITY

```python
#1 model
chain = Chain()
node_logit = PrimaryNode('logit')

node_lda = PrimaryNode('lda')
node_rf = SecondaryNode('rf')

node_rf.nodes_from = [node_logit, node_lda]

chain.add_node(node_rf)

chain.fit(train_data)
results = chain.predict(test_data)
#2 model
def get_simple_chain():
    first = PrimaryNode(model_type='logit')
    second = PrimaryNode(model_type='lda')
    final = SecondaryNode(model_type='rf',
                          nodes_from=[first, second])

    chain = Chain(final)

    return chain

#3 model
def get_simple_chain():
    first = PrimaryNode(model_type='xgboost')
    second = PrimaryNode(model_type='lda')
    final = SecondaryNode(model_type='rf',
                          nodes_from=[first, second])

    chain = Chain(final)

    return chain
```

Metric	1 model	2 model	3 model
Roc_auc_value	0.9892	0.9901	1.0
Precision	0.9971	0.9944	1.0
Recall	0.9010	0.9035	1.0
Accuracy	0.9998	0.9998	1.0

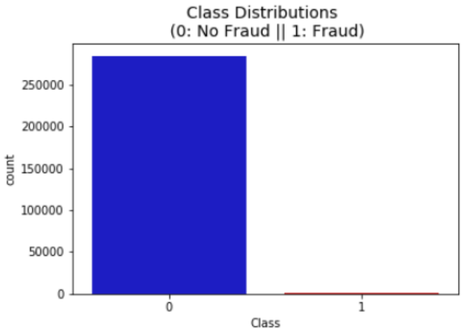
```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.90      0.99      0.94        91
           1       0.99      0.90      0.94        99

    accuracy                           0.94       190
   macro avg       0.94      0.94      0.94       190
weighted avg       0.95      0.94      0.94       190

KNears Neighbors:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93        91
           1       1.00      0.86      0.92        99

    accuracy                           0.93       190
   macro avg       0.93      0.93      0.93       190
weighted avg       0.94      0.93      0.93       190

Support Vector Classifier:
              precision    recall  f1-score   support

           0       0.88      0.99      0.93        91
           1       0.99      0.88      0.93        99

    accuracy                           0.93       190
   macro avg       0.94      0.93      0.93       190
weighted avg       0.94      0.93      0.93       190
```

Description of the dataset:

It is important that credit card companies can recognize fraudulent credit card transactions so that customers do not pay for goods they did not buy.

The main characteristics of the dataset:

Variables obtained with the help of PCA are the factors;
Dataset is not balanced;
The task of classification (detection of anomalies);
28 variables.



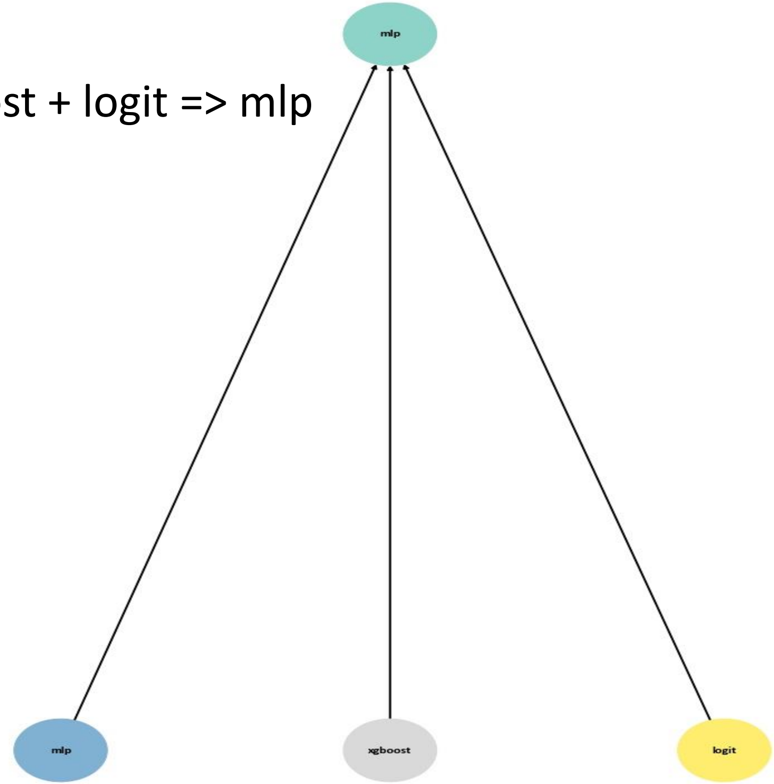
Composite model evaluation

ITMO UNIVERSITY

```
ROC AUC metric is 0.975
PRECISION metric is 0.9560439560439561
RECALL metric is 0.8877551020408163
ACCURACY metric is 0.9238578680203046
```

mlp + xgboost + logit => mlp

```
def get_model(train_file_path: str, cur_lead_time: datetime.timedelta = timedelta(minutes=10)):
    task = Task(task_type=TaskTypesEnum.classification)
    dataset_to_compose = InputData.from_csv(train_file_path, task=task)

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

    metric_function = MetricsRepository(). \
        metric_by_id(ClassificationMetricsEnum.ROCAUC_penalty)

    composer_requirements = GPComposerRequirements(
        primary=available_model_types, secondary=available_model_types,
        max_lead_time=cur_lead_time, max_arity=3,
        max_depth=4, pop_size=20, num_of_generations=100,
        crossover_prob = 0.8, mutation_prob = 0.8,
        add_single_model_chains = False)

    # Create the genetic programming-based composer, that allow to find
    # the optimal structure of the composite model
    composer = GPComposer()

    # run the search of best suitable model
    chain_evo_composed = composer.compose_chain(data=dataset_to_compose,
                                                initial_chain=None,
                                                composer_requirements=composer_requirements,
                                                metrics=metric_function, is_visualise=False)
    chain_evo_composed.fit(input_data=dataset_to_compose)

    return chain_evo_composed
```



Models comparison – code sinppet

```
def apply_model_to_data(model: Chain, data_path: str):
    """
    Applying model to data and check metrics.
    """
    dataset_to_validate = InputData.from_csv(data_path)

    predicted_labels = model.predict(dataset_to_validate).predict


    roc_auc_st =
round(roc_auc_score(y_true=dataset_to_validate.target,y_score=predicted_labels.round()), 4)

    p = round(precision_score(y_true=dataset_to_validate.target,y_pred=predicted_labels.round()), 4)
    r = round(recall_score(y_true=dataset_to_validate.target,y_pred=predicted_labels.round()), 4)
    a = round(accuracy_score(y_true=dataset_to_validate.target,y_pred=predicted_labels.round()),4 )
    f = round(f1_score(y_true=dataset_to_validate.target,y_pred=predicted_labels.round()), 4)

    return roc_auc_st, p, r, a, f
```

Models comparison - results

	SamplerUnder							
	roc_auc		precision		recall		accuracy	
	Kaggle	Result	Kaggle	Result	Kaggle	Result	Kaggle	Result
logit		0,969		0,956		0,897		0,928
lda		0,958		0,987		0,816		0,903
qda		0,961		0,935		0,887		0,913
dt		0,903		0,954		0,846		0,903
rf		0,978		0,946		0,897		0,923
mlp		0,963		0,936		0,897		0,918
knn		0,954		0,955		0,877		0,918
svc		0,964		0,956		0,897		0,928
xgboost		0,973		0,936		0,897		0,918
bernb		0,951		0,987		0,806		0,898
logit+lda=>rf		0,9595		0,9565		0,8979		0,9289

```
ROC AUC metric is 0.978
PRECISION metric is 0.967032967032967
RECALL metric is 0.8979591836734694
ACCURACY metric is 0.934010152284264
```

direct_data_model+ logit => rf

```
ROC AUC metric is 0.969
PRECISION metric is 0.9263157894736842
RECALL metric is 0.8979591836734694
ACCURACY metric is 0.9137055837563451
```

mlp

Data balancing

Balancing of the sample

```
def balance_class(file_path):
    """
    Function to balace our dataset to minority class.
    """
    file_name = file_path.replace('.', '/').split('/')[-2]

    df = pd.read_csv(file_path)

    X = df.drop(columns=['Class'])
    y = df.iloc[:,[-1]]

    rus = RandomUnderSampler(sampling_strategy = 'all', random_state=42)

    X_res, y_res = rus.fit_resample(X, y)
    X_res['Class'] = y_res

    df_balanced = shuffle(X_res, random_state = 42).reset_index().drop(columns='index')
    df_balanced.to_csv(fr'./{file_name}_underSample.csv', index=False)

    full_path = './' + file_name + '_underSample.csv'

    return full_path
```

Quality of single and composite models depending on the balance of the sample.

- Full Dataset

```
def get_simple_chain():
    first = PrimaryNode(model_type='logit')
    second = PrimaryNode(model_type='lda')
    final = SecondaryNode(model_type='rf',
                          nodes_from=[first, second])

    chain = Chain(final)

    return chain
```

```
ROC_AUC = 0.9027
PRECISION = 0.7245
RECALL = 0.7245
ACCURACY = 0.9991
f1_score = 0.7245
```

```
def get_simple_chain():
    first = PrimaryNode(model_type='mlp')

    chain = Chain(first)

    return chain
```

```
ROC_AUC = 0.9609
PRECISION = 0.9286
RECALL = 0.7959
ACCURACY = 0.9995
f1_score = 0.8571
```

Execution metrics and time

- X_train = 787

```
def get_simple_chain():
    first = PrimaryNode(model_type='knn')
    chain = Chain(first)

    return chain
```

```
file_path_first = r'./creditcard_scaling_underSample.csv'
```

```
train_file_path = r'./examples/data/creditcard_scaling_underSample/train.csv'
test_file_path = r'./examples/data/creditcard_scaling_underSample/test.csv'
```

```
train_data = InputData.from_csv(train_file_path)
test_data = InputData.from_csv(test_file_path)
```

```
chain = get_simple_chain()
```

```
start = time.time()
chain.fit(train_data, use_cache=False)
end = time.time()
print(end-start)
```

```
3.235996723175049
```

```
ROC_AUC = 0.9673
PRECISION = 0.9167
RECALL = 0.8851
ACCURACY = 0.9137
f1_score = 0.9006
```

```
ROC_AUC = 0.9716
PRECISION = 0.0296
RECALL = 0.8902
ACCURACY = 0.9493
f1_score = 0.0572
```

```
neigh = KNeighborsClassifier(n_neighbors=5)
```

```
start = time.time()
neigh.fit(X_train, y_train)
end = time.time()
print(end-start)
```

```
0.006994962692260742
```

```
ROC_AUC = 0.9268
PRECISION = 0.9294
RECALL = 0.908
ACCURACY = 0.9289
f1_score = 0.9186
```

```
ROC_AUC = 0.9317
PRECISION = 0.027
RECALL = 0.9207
ACCURACY = 0.9426
f1_score = 0.0525
```

Quality of single and composite models depending on the balance of the sample.

- underSample Dataset

```
def get_simple_chain():
    first = PrimaryNode(model_type='rf')
    second = PrimaryNode(model_type='svc')
    final = SecondaryNode(model_type='rf',
                          nodes_from=[first, second])

    chain = Chain(final)

    return chain
```

```
ROC_AUC = 0.9755
PRECISION = 0.9651
RECALL = 0.954
ACCURACY = 0.9645
f1_score = 0.9595
```

```
def get_simple_chain():
    first = PrimaryNode(model_type='mlp')

    chain = Chain(first)

    return chain
```

```
ROC_AUC = 0.9886
PRECISION = 0.9762
RECALL = 0.9425
ACCURACY = 0.9645
f1_score = 0.9591
```

```
composer_requirements = GPComposerRequirements(
    primary=available_model_types, secondary=available_model_types,
    max_lead_time=cur_lead_time, max_arity=3,
    max_depth=4, pop_size=20, num_of_generations=100,
    crossover_prob = 0.8, mutation_prob = 0.8,
    add_single_model_chains = True)
```

```
ROC_AUC_ALL = 0.9889
PRECISION = 0.9878
RECALL = 0.931
ACCURACY = 0.9645
F1_SCORE = 0.9586
```

logit+lda=>rf

Quality of single and composite models depending on the balance of the sample.

- Balance with the SamplerUnder
- 984 obs

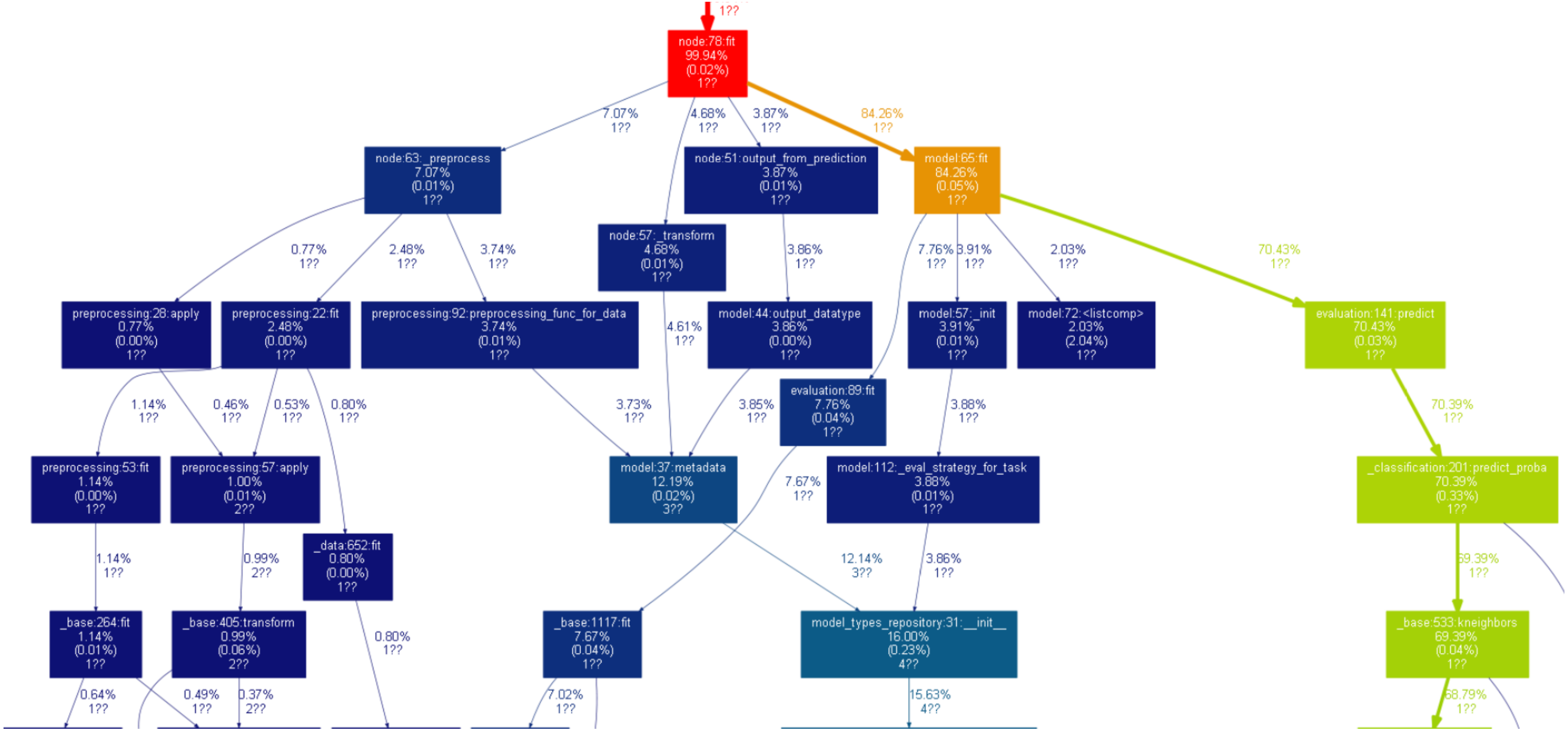
```
composer_requirements = GPComposerRequirements(
    primary=available_model_types, secondary=available_model_types,
    max_lead_time=cur_lead_time, max_arity=3,
    max_depth=4, pop_size=20, num_of_generations=100,
    crossover_prob = 0.8, mutation_prob = 0.8,
    add_single_model_chains = False)
```

	Under_samler			
	test_size = 0.2		test_size=0.3	
	Single	Compose	Single	Compose
roc_auc	0,969	0,96	0,971	0,971
precision	0,92631	0,94623	0,94326	0,9635
recall	0,89795	0,89795	0,91095	0,9041
accuracy	0,9137	0,92385	0,92905	0,93581
model_type	mlp	qda+direct_data_model => logit	mlp	svc+lda=>logit
fitness	0,992337	0,992678	0,990352	0,989685

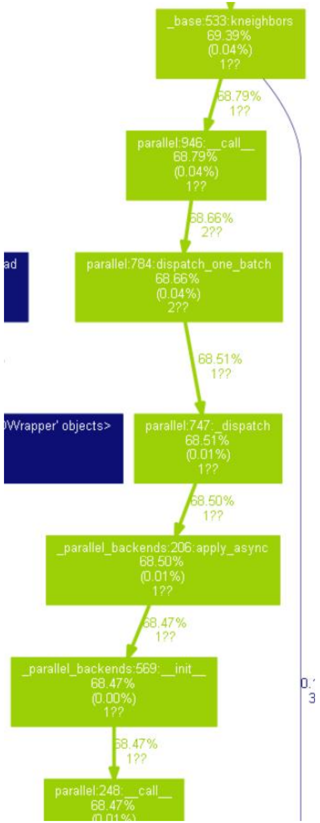
Performance analyzis

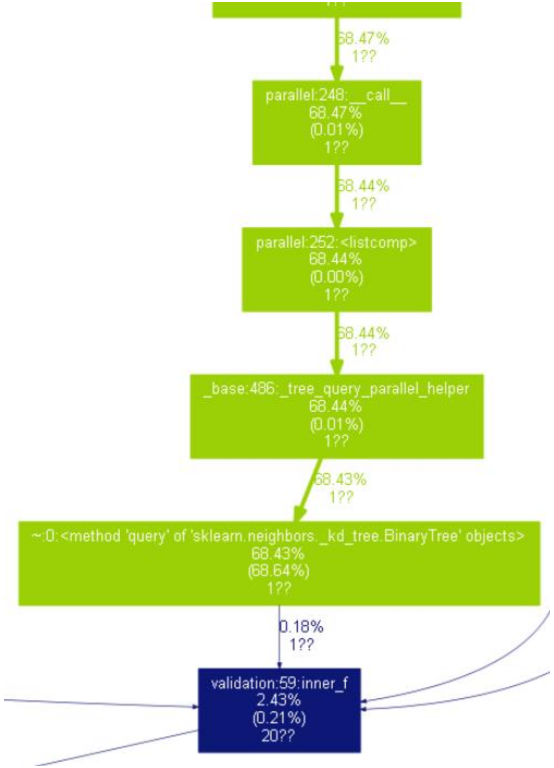
Results of knn profiling in Fedot

ITMO UNIVERSITY



Results of knn profiling in Fedot





Спасибо за внимание!

www.ifmo.ru

ITsMOre than a UNIVERSITY