ЛАБОРАТОРНАЯ РАБОТА 5

```
# -*- coding: utf-8 -*-
"""lab5.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1pYuU_UwiD49aoN4Mja5ZmJMMpbCd0PcM

# Лабораторная работа №5
"""

# Commented out IPython magic to ensure Python compatibility.
# %pip install -q keras_tuner pykan

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler, LabelBinarizer
from sklearn.neural_network import MLPClassifier
from sklearn.utils.class_weight import compute_class_weight
from sklearn.svm import SVC
from sklearn.model_selection import (train_test_split,
                                     GridSearchCV,
                                     RandomizedSearchCV,
                                     StratifiedKFold)
from sklearn.metrics import (classification_report,
                             confusion_matrix,
                             ConfusionMatrixDisplay,
                             RocCurveDisplay)

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.metrics import (Accuracy, F1Score,
                                      Precision, Recall, AUC)
from tensorflow.keras.optimizers import SGD, Adam

import keras_tuner as kt
from kan import KAN

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader

import random
random.seed(42)
```

```
np.random.seed(42)
torch.manual_seed(42)

import warnings
warnings.filterwarnings('ignore')

"""# Обработка датафрейма"""

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-red.csv'
df = pd.read_csv(url, sep=';')
df.head()

X = df.drop('quality', axis=1)
y = df['quality']

plt.figure(figsize=(8, 5))
ax = sns.countplot(x=y, palette='viridis', hue=y, legend=False)
plt.title('Распределение классов')
plt.xlabel('Оценка качества')
plt.ylabel('Количество экземпляров')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() /
2., p.get_height()),
                ha='center', va='center', xytext=(0, 5),
textcoords='offset points')
plt.show()

plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Корреляционная матрица признаков')
plt.show()

"""# Сплит и стандартизация датафрейма"""

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
stratify=y, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.375, stratify=y_train, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

classes = np.unique(y_train)
class_to_idx = {c: i for i, c in enumerate(classes)}
y_train = np.array([class_to_idx[c] for c in y_train])
y_val = np.array([class_to_idx[c] for c in y_val])
y_test = np.array([class_to_idx[c] for c in y_test])
```

```
num_classes = 6
num_features = X_train.shape[1]

scoring = {
    "accuracy": "accuracy",
    "f1_macro": "f1_macro",
    "precision_macro": "precision_macro",
    "recall_macro": "recall_macro",
    "roc_auc_ovr": "roc_auc_ovr"
}

label_binarizer = LabelBinarizer()
y_onehot_train = label_binarizer.fit_transform(y_train)
y_onehot_val = label_binarizer.transform(y_val)
y_onehot_test = label_binarizer.transform(y_test)

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

"""# Обучение моделей

## MLP

- `hidden_layer_sizes`: отвечает за архитектуру сети. Мы выбираем количество нейронов в скрытых слоях и их количество
- `activation`: функция активации после выходов скрытых слоев. Важно! Между последним скрытым слоем и выходом нейросети библиотека сама ставит либо sigmoid (для бинарной классификации), либо softmax (мультиклассовая классификация). То есть нейросеть возвращает не логиты, а "вероятности"
- `solver`: алгоритм обновления весов модели. Здесь выбраны стохастический градиентный спуск (sgd) и adam (продвинутая версия с моментами)
- `alpha`: параметр регуляризации L2
- `learning_rate`: скорость обучения. Выбраны два: constant и invscaling. Первый соответственно не меняется, второй уменьшает скорость обучения
"""

# Commented out IPython magic to ensure Python compatibility.
# %%time
#
# param_mlp = {
#     'hidden_layer_sizes': [(50,50,50), (100,100,100), (50,100,50), (100,)],
#     'activation': ['relu', 'logistic'],
#     'solver': ['sgd', 'adam'],
#     'alpha': [0.0001, 0.05],
#     'learning_rate': ['constant', 'invscaling'],
# }
#
# mlp = GridSearchCV(
#     MLPClassifier(random_state=42),
```

```
#     param_grid=param_mlp,
#     scoring=scoring,
#     refit="f1_macro",
#     cv=cv,
#     n_jobs=-1,
#     verbose=1
# )
# mlp.fit(X_train, y_train)

print(f"Лучшие параметры модели {mlp.best_params_}")

"""### Метрики на обучающей выборке"""

metrics_df = pd.DataFrame(mlp.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_f1_macro',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

y_score = mlp.predict_proba(X_train)

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_train[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

y_pred = mlp.predict(X_train)
ConfusionMatrixDisplay.from_predictions(y_train, y_pred, cmap='Blues')
plt.title("Train confusion matrix")
plt.show()
```

```
"""### Метрики на валидационной выборке"""

y_pred = mlp.predict(X_val)
print(classification_report(y_val, y_pred))

y_score = mlp.predict_proba(X_val)

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_val[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_val, y_pred, cmap='Blues')
plt.title("Validation confusion matrix")
plt.show()

"""## SVM"""

svm_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto', 1, 10, 0.1, 0.01],
    'coef0': [0, 1, 2, 5, 10, 0.1, 0.01],
    'degree': [2, 3, 4, 5],
    'C': [0.01, 0.1, 0.5, 1, 2, 10, 20]
}

svm = RandomizedSearchCV(
    estimator=SVC(probability=True, random_state=42),
    param_distributions=svm_grid,
    scoring=scoring,
    refit="f1_macro",
    n_jobs=-1,
    cv=cv,
    random_state=42,
    verbose=1
)
svm.fit(X_train, y_train)
```

```
print(f"Лучшие параметры модели {svm.best_params_}")

"""### Метрики на обучающей выборке"""

metrics_df = pd.DataFrame(svm.cv_results_)[[
    'mean_test_accuracy',
    'mean_test_precision_macro',
    'mean_test_recall_macro',
    'mean_test_f1_macro',
    'mean_test_roc_auc_ovr'
]]

metrics_df = metrics_df.sort_values(by='mean_test_f1_macro',
ascending=False).iloc[0, :]
metrics_df.name = 'Metric'

display(metrics_df)

y_score = svm.predict_proba(X_train)

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_train[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

y_pred = svm.predict(X_train)
ConfusionMatrixDisplay.from_predictions(y_train, y_pred, cmap='Blues')
plt.title("Train confusion matrix")
plt.show()

"""### Метрики на валидационной выборке"""

y_pred = svm.predict(X_val)
print(classification_report(y_val, y_pred))

y_score = svm.predict_proba(X_val)
```

```
fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_val[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_val, y_pred, cmap='Blues')
plt.title("Validation confusion matrix")
plt.show()

"""## MLP Keras"""

def build_model(hp):
    model = Sequential()
    structure = hp.Choice('structure', values=['3_layers_50',
'3_layers_100', 'pyramid', '1_layer_100'])

    if structure == '3_layers_50':
        layers_list = [50, 50, 50]
    elif structure == '3_layers_100':
        layers_list = [100, 100, 100]
    elif structure == 'pyramid':
        layers_list = [50, 100, 50]
    else:
        layers_list = [100]

    activation_choice = hp.Choice('activation', values=['relu',
'sigmoid'])

    for units in layers_list:
        model.add(Dense(units=units, activation=activation_choice))
        model.add(Dropout(0.2))

    model.add(Dense(num_classes, activation='softmax'))

    solver = hp.Choice('solver', values=['sgd', 'adam'])
    alpha = hp.Float('alpha', min_value=1e-4, max_value=0.05,
sampling='log')
```

```
    if solver == 'adam':
        optimizer = Adam(learning_rate=1e-3, weight_decay=alpha)
    else:
        optimizer = SGD(learning_rate=1e-2, weight_decay=alpha)

    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=[
            'accuracy',
            F1Score(average='macro', name='f1_macro'),
            Precision(name='precision'),
            Recall(name='recall'),
            AUC(name='auc_ovr', multi_label=False)
        ]
    )
    return model

tuner = kt.RandomSearch(
    build_model,
    objective=kt.Objective("val_f1_macro", direction="max"),
    max_trials=20,
    executions_per_trial=1
)

tuner.search(X_train, y_onehot_train, epochs=100,
validation_data=(X_val, y_onehot_val))
mlp_keras = tuner.get_best_models(num_models=1)[0]

pd.DataFrame(tuner.get_best_hyperparameters()[0].values,
index=['value']).T

"""### Метрики на обучающей выборке"""

y_pred = np.argmax(mlp_keras.predict(X_train), axis=1)
y_true = np.argmax(y_onehot_train, axis=1)
print(classification_report(y_true, y_pred))

y_score = mlp_keras.predict(X_train)

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_train[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
```

```
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
plt.title("Train confusion matrix")
plt.show()

"""### Метрики на валидационной выборке"""

y_pred = np.argmax(mlp_keras.predict(X_val), axis=1)
y_true = np.argmax(y_onehot_val, axis=1)
print(classification_report(y_true, y_pred))

y_score = mlp_keras.predict(X_val)

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_val[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
plt.title("Validation confusion matrix")
plt.show()

"""## KAN

Kolmogorov-Arnold Networks — это альтернатива привычным многослойным
перцептронам (MLP).

Главное отличие: в MLP функции активации (например, ReLU) закреплены в
нейронах, а веса — это числа на связях. В KAN всё наоборот: на связях
стоят обучаемые функции, а узлы просто суммируют их результаты
```

```
- `width`: входной слой, один скрытый слой и выходной слой. Скрытый слой здесь означает не количество нейронов с активацией, а количество сумматоров. Между слоями будут обучаемые одномерные функции.
- `grid`: каждая функция на ребре — это B-сплайн
- `k`: степень B-сплайна

Каждое ребро в KAN вычисляет: $f(x) = f_{base}(x) + f_{spline}(x)$

Base function: Обычно это $SiLU(x) = \frac{x}{1+e^{-x}}$, чтобы даже при нулевом сплайне сохранялся градиент.

Spline: Адаптивная часть, которая подстраивается под данные, меняя свою форму во время обучения.
"""

X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.long)

X_val = torch.tensor(X_val, dtype=torch.float32)
y_val = torch.tensor(y_val, dtype=torch.long)

X_test = torch.tensor(X_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.long)

kan = KAN(
    width=[num_features, 64, num_classes],
    grid=10,
    k=3
)

dataset = {
    'train_input': X_train,
    'train_label': y_train,
    'test_input': X_val,
    'test_label': y_val
}

results = kan.fit(
    dataset,
    opt="Adam",
    steps=500,
    update_grid=True,
    grid_update_num=20,
    loss_fn=nn.CrossEntropyLoss(),
    lr=1e-2
)

x = torch.normal(0, 1, size=(100,11))
kan(x)
kan.plot(scale=0.5)
```

```
"""### Метрики на обучающей выборке


"""

y_pred = torch.argmax(kan(X_train), dim=1)
y_true = np.argmax(y_onehot_train, axis=1)
print(classification_report(y_true, y_pred))

logits = kan(X_train)
y_score = F.softmax(logits, dim=1).detach().numpy()

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_train[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
plt.title("Train confusion matrix")
plt.show()

"""### Метрики на валидационной выборке

"""

y_pred = torch.argmax(kan(X_val), dim=1)
y_true = np.argmax(y_onehot_val, axis=1)
print(classification_report(y_true, y_pred))

logits = kan(X_val)
y_score = F.softmax(logits, dim=1).detach().numpy()

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_val[:, i],
        y_score[:, i],
```

```
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
plt.title("Validation confusion matrix")
plt.show()

"""# *MLP PyTorch"""

class MLP(nn.Module):
    def __init__(self, input_dim=num_features, output_dim=num_classes):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(0.3),

            nn.Linear(64, 128),
            nn.BatchNorm1d(128),
            nn.ReLU(),
            nn.Dropout(0.3),

            nn.Linear(128, 64),
            nn.BatchNorm1d(64),
            nn.ReLU(),
            nn.Dropout(0.3),

            nn.Linear(64, output_dim)
        )

    def forward(self, x):
        return self.model(x)

mlp_torch = MLP()
optimizer = torch.optim.Adam(mlp_torch.parameters(), lr=1e-2,
weight_decay=1e-3)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode='min', factor=0.5, patience=10)

train_ds = TensorDataset(X_train, y_train)
val_ds = TensorDataset(X_val, y_val)
```

```
train_loader = DataLoader(train_ds, batch_size=32, shuffle=True)
val_loader = DataLoader(val_ds, batch_size=32, shuffle=False)

loss_fn = nn.CrossEntropyLoss()

for epoch in range(100):
    mlp_torch.train()
    total_train_loss = 0

    for X_batch, y_batch in train_loader:
        pred = mlp_torch(X_batch)
        loss = loss_fn(pred, y_batch)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_train_loss += loss.item() * X_batch.size(0)

    mlp_torch.eval()
    total_val_loss = 0
    with torch.no_grad():
        for X_batch, y_batch in val_loader:
            val_pred = mlp_torch(X_batch)
            val_loss = loss_fn(val_pred, y_batch)
            total_val_loss += val_loss.item() * X_batch.size(0)

    avg_train_loss = total_train_loss / len(train_ds)
    avg_val_loss = total_val_loss / len(val_ds)

    scheduler.step(avg_val_loss)

    if (epoch + 1) % 10 == 0:
        print(f"Epoch {epoch+1:3d}/100 | Train Loss:
{avg_train_loss:.5f} | Val Loss: {avg_val_loss:.5f}")

"""### Метрики на обучающей выборке

"""

y_pred = torch.argmax(mlp_torch(X_train), dim=1)
y_true = np.argmax(y_onehot_train, axis=1)
print(classification_report(y_true, y_pred))

logits = mlp_torch(X_train)
y_score = F.softmax(logits, dim=1).detach().numpy()

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
```

```
    RocCurveDisplay.from_predictions(
        y_onehot_train[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
plt.title("Train confusion matrix")
plt.show()

"""### Метрики на валидационной выборке

"""

y_pred = torch.argmax(mlp_torch(X_val), dim=1)
y_true = np.argmax(y_onehot_val, axis=1)
print(classification_report(y_true, y_pred))

logits = mlp_torch(X_val)
y_score = F.softmax(logits, dim=1).detach().numpy()

fig, ax = plt.subplots(figsize=(12, 8))

for i, class_name in enumerate(label_binarizer.classes_):
    RocCurveDisplay.from_predictions(
        y_onehot_val[:, i],
        y_score[:, i],
        name=f"Class {class_name}",
        ax=ax,
    )

plt.plot([0, 1], [0, 1], "k--", label="Random prediction (AUC = 0.5)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC curves (One-vs-Rest)")
plt.legend()
plt.tight_layout()
plt.show()

ConfusionMatrixDisplay.from_predictions(y_true, y_pred, cmap='Blues')
plt.title("Validation confusion matrix")
plt.show()
```

```
"""# Сравнение моделей"""

models = {
    "mlp": mlp,
    "svm": svm,
    "mlp_keras": mlp_keras,
    "kan": kan,
    "mlp_torch": mlp_torch
}

results = []
for name, model in models.items():
    if name in {'mlp', 'svm'}:
        X_test_np = X_test if isinstance(X_test, np.ndarray) else
X_test.numpy()
        y_pred = model.predict(X_test_np)
        score = classification_report(y_test, y_pred, output_dict=True)
    elif name == 'mlp_keras':
        X_test_np = X_test if isinstance(X_test, np.ndarray) else
X_test.numpy()
        y_pred = np.argmax(model(X_test_np), axis=1)
        y_true = np.argmax(y_onehot_test, axis=1)
        score = classification_report(y_true, y_pred, output_dict=True)
    else:
        X_test_torch = X_test if isinstance(X_test, torch.Tensor) else
torch.tensor(X_test, dtype=torch.float32)
        y_pred = torch.argmax(model(X_test_torch), dim=1).cpu().numpy()
        y_true = np.argmax(y_onehot_test, axis=1)
        score = classification_report(y_true, y_pred, output_dict=True)

    results.append({
        "model": name,
        "accuracy": score["accuracy"],
        "f1_score": score["macro avg"]["f1-score"],
        "precision": score["macro avg"]["precision"],
        "recall": score["macro avg"]["recall"],
    })

pd.DataFrame(results).sort_values(by="f1_score",
ascending=False).reset_index(drop=True)

"""# Вывод по лабораторной работе

В ходе работы было обучено и сравнено пять моделей классификации на
наборе данных о качестве красного вина (задача многоклассовой
классификации, 6 классов, несбалансированные данные). Использовались
стандартизация, стратифицированная кросс-валидация и единые
обучающая/валидационная/тестовая выборки.

### 1. MLP (sklearn, GridSearchCV)
```

```
- **Лучшие параметры**: `activation='relu'`, `alpha=0.05`,
`hidden_layer_sizes=(50,100,50)`, `solver='adam'`
- **Кросс-валидация (обучение)**: Accuracy = 0.591, F1 (macro) = 0.302,
ROC AUC = 0.771
- **Валидация**: Accuracy = 0.61, F1 (macro) = 0.43, Precision (macro) =
0.41, Recall (macro) = 0.45

Модель показывает стабильные, но умеренные результаты. Переобучения нет,
ROC AUC высокий.

### 2. SVM (RandomizedSearchCV)
- **Лучшие параметры**: `kernel='poly'`, `gamma='auto'`, `degree=4`,
`coef0=5`, `C=0.5`
- **Кросс-валидация (обучение)**: Accuracy = 0.573, F1 (macro) = 0.332,
ROC AUC = 0.799
- **Валидация**: Accuracy = 0.60, F1 (macro) = 0.38, Precision (macro) =
0.37, Recall (macro) = 0.42

SVM показывает результаты, близкие к MLP, но с немного более низким F1.

### 3. KAN (Kolmogorov–Arnold Network)
- **Архитектура**: [11 → 64 → 6], grid=10, k=3
- **Обучение**: Accuracy = 1.00, F1 (macro) = 1.00 (полное запоминание)
- **Валидация**: Accuracy = 0.65, F1 (macro) = 0.47, Precision (macro) =
0.51, Recall (macro) = 0.45

KAN полностью переобучился (100% на трейне), но на валидации показывает
лучшую точность (0.65) и F1 (0.47) среди всех моделей. Требуется
регуляризация.

### 4. MLP (Keras, KerasTuner)
- **Лучшие параметры**: `structure='3_layers_100'`, `activation='relu'`,
`solver='adam'`, `alpha≈0.0449`
- **Обучение**: точность высокая (по графикам F1 ~0.87 на трейне)
- **Валидация**: Accuracy = 0.63, F1 (macro) = 0.45, Precision (macro) =
0.45, Recall (macro) = 0.48

Одна из лучших моделей по балансу метрик на валидации. Гибкость
настройки (dropout, оптимизатор) даёт преимущество.

### 5. MLP (PyTorch, BatchNorm + Dropout)
- **Архитектура**: 64 → 128 → 64 → 6 (ReLU, Dropout 0.3)
- **Обучение**: Accuracy = 0.71, F1 (macro) = 0.49
- **Валидация**: Accuracy = 0.61, F1 (macro) = 0.35, Precision (macro) =
0.34, Recall (macro) = 0.36

Модель заметно слабее Keras-версии. Возможные причины: неудачная
архитектура, недостаточная настройка гиперпараметров, отсутствие
балансировки классов.

---
```

```
## Итоговое сравнение моделей по валидации (фактические метрики)

| Модель         | Accuracy | F1 (macro) | Precision (macro) | Recall
(macro) |
|----------------|----------|------------|-------------------|---------
-------|
| MLP (sklearn)  | 0.61     | 0.43       | 0.41              |
0.45           |
| SVM            | 0.60     | 0.38       | 0.37              |
0.42           |
| KAN            | **0.65** | **0.47**   | **0.51**          |
0.45           |
| MLP (Keras)    | 0.63     | 0.45       | 0.45              |
**0.48**       |
| MLP (PyTorch)  | 0.61     | 0.35       | 0.34              |
0.36           |

---

## Общее заключение

1. **Лучшая точность (Accuracy) на валидации** — у **KAN** (0.65), но
модель сильно переобучена.
2. **Лучший баланс Precision/Recall/F1** — у **MLP (Keras)**, которая
показывает стабильные результаты без явного переобучения.
3. **SVM** и **MLP (PyTorch)** уступают двум лучшим моделям по
большинству метрик.
4. Все модели испытывают трудности с **редкими классами** (0, 1, 5) из-
за сильного дисбаланса данных.

**Рекомендации**:
- Применить взвешивание классов или SMOTE для борьбы с дисбалансом.
- Для KAN — добавить раннюю остановку и регуляризацию.
- Для MLP (PyTorch) — пересмотреть архитектуру и провести подбор
гиперпараметров.

**Практический вывод**: на данном наборе данных **MLP (Keras)** является
наиболее сбалансированной и готовой к использованию моделью, а **KAN** —
перспективной, но требующей доработки.
"""
```