МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт компьютерных наук
Кафедра автоматизированных систем управления

Лабораторная работа №2

по дисциплине «Машинное обучение»

Студент М-РИТ-25-1 _____ Станиславчук С. М.

(подпись, дата)

Руководитель

Доцент, д.т.н. _____ Сараев П. В.

(подпись, дата)

Липецк 2025

1. Задание кафедры

1) Загрузите базу данных Iris.
2) Выполните задания 2 — 10 из методич. указаний № 571 для л.р.2

2. Цель работы

Цель работы — изучение алгоритмов построения классификаторов, оценки качества классификации и средств классификации данных в языке R / Python.

3. Ход работы

В качестве языка программирования для выполнения лабораторной работы, я выбрал Python.

Для исследования влияния аргументов функций, реализующий методы классификации, на точность классификации были выбраны следующие комбинации:

```
# --- DecisionTreeClassifier ---
max_depth_values = [1, 2, 3, 10]
criterion_values = ["gini", "entropy"]

# --- SVM ---
kernels = ["linear", "rbf"]
C_values = [0.1, 1, 10]
gamma_values = ["scale", "auto"]

# --- GaussianNB ---
var_smoothing_values = [1e-9, 1e-5, 1e-2, 0, 0.1, 1]
```

Пример выполнения программы №8:

Model	Params	Accuracy
DecisionTree	max_depth=2, criterion=gini	0.933333
DecisionTree	max_depth=2, criterion=entropy	0.933333
DecisionTree	max_depth=10, criterion=gini	0.900000
DecisionTree	max_depth=10, criterion=entropy	0.900000
DecisionTree	max_depth=3, criterion=gini	0.866667
DecisionTree	max_depth=3, criterion=entropy	0.866667
DecisionTree	max_depth=1, criterion=gini	0.700000
DecisionTree	max_depth=1, criterion=entropy	0.700000
GaussianNB	var_smoothing=0.01	1.000000
GaussianNB	var_smoothing=1e-09	0.966667
GaussianNB	var_smoothing=1e-05	0.966667
GaussianNB	var_smoothing=0	0.966667
GaussianNB	var_smoothing=0.1	0.900000
GaussianNB	var_smoothing=1	0.900000
SVM	kernel=linear, C=1, gamma=scale	1.000000
SVM	kernel=linear, C=1, gamma=auto	1.000000
SVM	kernel=linear, C=2, gamma=scale	1.000000
SVM	kernel=linear, C=2, gamma=auto	1.000000
SVM	kernel=linear, C=5, gamma=scale	1.000000
SVM	kernel=linear, C=5, gamma=auto	1.000000
SVM	kernel=linear, C=10, gamma=scale	1.000000
SVM	kernel=linear, C=10, gamma=auto	1.000000
SVM	kernel=rbf, C=1, gamma=auto	1.000000
SVM	kernel=rbf, C=5, gamma=scale	1.000000
SVM	kernel=rbf, C=5, gamma=auto	1.000000
SVM	kernel=rbf, C=10, gamma=scale	1.000000
SVM	kernel=rbf, C=10, gamma=auto	1.000000
SVM	kernel=rbf, C=2, gamma=scale	0.966667
SVM	kernel=rbf, C=2, gamma=auto	0.966667
SVM	kernel=linear, C=0.1, gamma=scale	0.933333
SVM	kernel=linear, C=0.1, gamma=auto	0.933333
SVM	kernel=rbf, C=0.1, gamma=auto	0.933333
SVM	kernel=rbf, C=1, gamma=scale	0.933333
SVM	kernel=rbf, C=0.1, gamma=scale	0.900000

Анализ.

1. DTC: слишком малая глубина приводит к большим неточностям, глубину лучше брать >= 2. В качестве criterion для тест. значений выбор не влиял на точность (из-за маленькой выборки). Вывод: глубина ~3, в качестве `criterion=entropy` (так как показывает на 1% лучше при анализе с большими (обучаемыми данными))

2. GaussianNB: `var_smoothing` регулирует «размытие» вероятностей. Слишком маленькое — почти не меняет модель, слишком большое — снижает точность. Лучше всего подходит значение 0.01

3. SVM: при малом C=0.1 получаем сильную регуляризацию, в результате точность падает. При C>=1, линейный SVM полностью подгоняет train; у RBF более постепенное увеличение, и только при C>=5 1.0. `gamma` для RBF kernel влияет на «скалирование» ядра, но на train точность почти не меняется при больших C.

3. Вывод

В результате выполненной лабораторной работы, изучил основные алгоритмы построения классификаторов и оценки качества классификации.

Приложение 1

```
# precision — точность (сколько объектов, отнесённых к этому классу, действительно принадлежат
ему).
# recall (sensitivity) — полнота (сколько объектов данного класса модель нашла).
# f1-score — гармоническое среднее между precision и recall.
# support — количество реальных объектов этого класса в тестовой выборке.

# separator
def print_separator():
    print('' + '-'*80)

random_state = 43

# 1. Загрузите с сайта UICI Machine Learning Repository базу данных
# Iris. В этой базе представлена информация о классификации цветов (ирисов)
# на 3 вида: Setosa, Versicolor и Virginica в зависимости от 4 параметров
# (sepal length in cm, sepal width in cm, petal length in cm, petal width in cm).

import pandas as pd

# fetch dataset
iris = pd.read_csv('iris.csv')

# print dataframe
print(iris)

# Output:
#      sepal length  sepal width  petal length  petal width          target
# 0             5.1          3.5           1.4          0.2     Iris-setosa
# 1             4.9          3.0           1.4          0.2     Iris-setosa
# 2             4.7          3.2           1.3          0.2     Iris-setosa
# 3             4.6          3.1           1.5          0.2     Iris-setosa
# 4             5.0          3.6           1.4          0.2     Iris-setosa
# ..            ...          ...           ...          ...             ...
# 145           6.7          3.0           5.2          2.3  Iris-virginica
# 146           6.3          2.5           5.0          1.9  Iris-virginica
# 147           6.5          3.0           5.2          2.0  Iris-virginica
# 148           6.2          3.4           5.4          2.3  Iris-virginica
# 149           5.9          3.0           5.1          1.8  Iris-virginica

print_separator()

# 2. Разделите данные на обучающую (80%) и тестовую (20%) множества. Выведите
# сводную информацию об обучающем множестве (1) количество строк во множестве,
# 2) количество объектов в каждом классе; 3) количество пропущенных значений;
# 4) минимальные, максимальные и средние значения по каждому из 4 параметров).

from sklearn.model_selection import train_test_split

# split dataset into training and testing sets
# random_state is set for reproducibility
train, test = train_test_split(iris, test_size=0.2, random_state=random_state)
```

```
# print training set info
print("\nTraining Set Info:")
# 1)
print(f"Number of rows: {train.shape[0]}")
# 2)
print("\nObjects per class:")
print(train['target'].value_counts())
# 3)
print("\nMissing values:")
print(train.isnull().sum())
# 4)
print("\nStatistical summary:")
print(train.describe().loc[['min', 'max', 'mean']])

print_separator()
# 3. Проведите классификациб данных обучабшего множество с помощью
# наивного байесовского классификатора.

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# encode target labels
le = LabelEncoder()
train['target'] = le.fit_transform(train['target'])
test['target'] = le.transform(test['target'])

print("\nEncoded Training Set:")
print(train)

# 4. Постройте таблицы классификации для обучающего и тестового
# множеств. Рассчитайте характеристики классификации (точность классификации,
# уровень ошибки; точность, чувствительность(recall) и специфичность(specificity)
# для каждого класса).

import numpy as np

# separate features and target
X_train = train.drop('target', axis=1)
y_train = train['target']
X_test = test.drop('target', axis=1)
y_test = test['target']

# initialize and train classifier
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# make predictions
y_train_pred = gnb.predict(X_train)
y_test_pred = gnb.predict(X_test)

# print classification report and confusion matrix for training set
print("\nTraining Set Classification Report:")
print(classification_report(y_train, y_train_pred, target_names=le.classes_))
print("\nTraining Set Confusion Matrix:")
print(confusion_matrix(y_train, y_train_pred))

def classification_metrics(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    classes = np.unique(y_true)

    metrics = {}
    for i, cls in enumerate(classes):
        tp = cm[i, i]
        fn = cm[i, :].sum() - tp
        fp = cm[:, i].sum() - tp
        tn = cm.sum() - (tp + fn + fp)

        accuracy = (tp + tn) / cm.sum()
```

```
        error_rate = 1 - accuracy
        sensitivity = tp / (tp + fn) if (tp + fn) > 0 else 0
        specificity = tn / (tn + fp) if (tn + fp) > 0 else 0

        metrics[cls] = {
            "accuracy": accuracy,
            "error_rate": error_rate,
            "sensitivity": sensitivity,
            "specificity": specificity
        }
    return metrics

print("\nTraining Set Metrics:")
for cls, m in classification_metrics(y_train, y_train_pred).items():
    print(f"Class {cls}: "
          f"Acc={m['accuracy']:.3f}, "
          f"Err={m['error_rate']:.3f}, "
          f"Sens={m['sensitivity']:.3f}, "
          f"Spec={m['specificity']:.3f}")

print("\nTest Set Metrics:")
for cls, m in classification_metrics(y_test, y_test_pred).items():
    print(f"Class {cls}: "
          f"Acc={m['accuracy']:.3f}, "
          f"Err={m['error_rate']:.3f}, "
          f"Sens={m['sensitivity']:.3f}, "
          f"Spec={m['specificity']:.3f}")

print_separator()


# 5. Проделайте шаги 3 и 4, только с учетом двух параметров (sepal length и sepal width).

# select only sepal length and sepal width
X_train_2d = X_train[['sepal length', 'sepal width']]
X_test_2d = X_test[['sepal length', 'sepal width']]

# initialize and train classifier
gnb_2d = GaussianNB()
gnb_2d.fit(X_train_2d, y_train)

# make predictions
y_train_pred_2d = gnb_2d.predict(X_train_2d)
y_test_pred_2d = gnb_2d.predict(X_test_2d)

# print classification report and confusion matrix for training set
print("\nTraining Set (2D) Classification Report:")
print(classification_report(y_train, y_train_pred_2d, target_names=le.classes_))
print("\nTraining Set (2D) Confusion Matrix:")
print(confusion_matrix(y_train, y_train_pred_2d))

print("\nTrain Set Metrics (2D):")
for cls, m in classification_metrics(y_train, y_train_pred_2d).items():
    print(f"Class {cls}: "
          f"Acc={m['accuracy']:.3f}, "
          f"Err={m['error_rate']:.3f}, "
          f"Sens={m['sensitivity']:.3f}, "
          f"Spec={m['specificity']:.3f}")

print("\nTest Set Metrics (2D):")
for cls, m in classification_metrics(y_test, y_test_pred_2d).items():
    print(f"Class {cls}: "
          f"Acc={m['accuracy']:.3f}, "
          f"Err={m['error_rate']:.3f}, "
          f"Sens={m['sensitivity']:.3f}, "
          f"Spec={m['specificity']:.3f}")


print_separator()
```

```
# 6. Постройте модели классификации с помощью деревьев решений для
# 4 (sepal length) и 2 (sepal width) параметров. Представьте модели
# классификации с помощью деревьев решений графически.

print('-'*12 + ' 6 ' + '-'*12)

from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
import matplotlib.pyplot as plt

# initialize and train classifier for 4 parameters
dtc = DecisionTreeClassifier(max_depth=10, random_state=random_state, criterion="gini")
X_train_2d_4_2 = X_train[['sepal length', 'sepal width']]
dtc.fit(X_train_2d_4_2, y_train)

X_test_2d_4_2 = X_test[['sepal length', 'sepal width']]

# make predictions
y_train_pred_dtc = dtc.predict(X_train_2d_4_2)
y_test_pred_dtc = dtc.predict(X_test_2d_4_2)

# print classification report and confusion matrix for training set
print("\nTraining Set (DTC) Classification Report:")
print(classification_report(y_train, y_train_pred_dtc, target_names=le.classes_))
print("\nTraining Set (DTC) Confusion Matrix:")
print(confusion_matrix(y_train, y_train_pred_dtc))

# Test classification report and confusion matrix
print("\nTest Set (DTC) Classification Report:")
print(classification_report(y_test, y_test_pred_dtc, target_names=le.classes_))
print("\nTest Set (DTC) Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred_dtc))
print("\nTrain Set Metrics (DTC):")
for cls, m in classification_metrics(y_train, y_train_pred_dtc).items():
    print(f"Class {cls}: "
          f"Acc={m['accuracy']:.3f}, "
          f"Err={m['error_rate']:.3f}, "
          f"Sens={m['sensitivity']:.3f}, "
          f"Spec={m['specificity']:.3f}")

print("\nTest Set Metrics (DTC):")
for cls, m in classification_metrics(y_test, y_test_pred_dtc).items():
    print(f"Class {cls}: "
          f"Acc={m['accuracy']:.3f}, "
          f"Err={m['error_rate']:.3f}, "
          f"Sens={m['sensitivity']:.3f}, "
          f"Spec={m['specificity']:.3f}")

x_min, x_max = X_train_2d_4_2['sepal length'].min() - 1, X_train_2d_4_2['sepal length'].max() + 1
y_min, y_max = X_train_2d_4_2['sepal width'].min() - 1, X_train_2d_4_2['sepal width'].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))

Z = dtc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# plt.figure(figsize=(8,6))
# plt.contourf(xx, yy, Z, alpha=0.3)
# plt.scatter(X_train_2d_4_2['sepal length'], X_train_2d_4_2['sepal width'],
#             c=y_train, edgecolor='k', cmap=plt.cm.Set1)
# plt.xlabel("sepal length")
# plt.ylabel("sepal width")
# plt.title("Decision boundaries (Decision Tree, 2 features)")
# plt.show()

print_separator()

# 7. Постройте модели классификации с помощью метода опорных векторов
```

```
# (SVM) для sepal length и sepal width параметров
from sklearn.svm import SVC

X_train_svm = X_train[['sepal length', 'sepal width']]
X_test_svm = X_test[['sepal length', 'sepal width']]

# teach SVM with RBF core
svm = SVC(kernel='rbf', gamma='auto', random_state=random_state)
svm.fit(X_train_svm, y_train)

y_train_pred_svm = svm.predict(X_train_svm)
y_test_pred_svm = svm.predict(X_test_svm)

print("\nTraining Set (SVM) Classification Report:")
print(classification_report(y_train, y_train_pred_svm, target_names=le.classes_))
print("\nTraining Set (SVM) Confusion Matrix:")
print(confusion_matrix(y_train, y_train_pred_svm))

print("\nTest Set (SVM) Classification Report:")
print(classification_report(y_test, y_test_pred_svm, target_names=le.classes_))
print("\nTest Set (SVM) Confusion Matrix:")
print(confusion_matrix(y_test, y_test_pred_svm))


# 8. Изучите и исследуйте влияние аргументов функций, реализующих методы
# классификации на точность классификации

# 9. Сведите результаты всех исследований в таблицу и сравните результаты классификации
# на основе оценки точности по тестовому множетсву

import itertools
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

results = []

# --- GaussianNB ---
var_smoothing_values = [1e-9, 1e-5, 1e-2, 0, 0.1, 1]
for vs in var_smoothing_values:
    model = GaussianNB(var_smoothing=vs)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results.append({
        "Model": "GaussianNB",
        "Params": f"var_smoothing={vs}",
        "Accuracy": acc
    })

# --- DecisionTreeClassifier ---
max_depth_values = [1, 2, 3, 10]
criterion_values = ["gini", "entropy"]
for md, cr in itertools.product(max_depth_values, criterion_values):
    model = DecisionTreeClassifier(max_depth=md, criterion=cr, random_state=random_state)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results.append({
        "Model": "DecisionTree",
        "Params": f"max_depth={md}, criterion={cr}",
        "Accuracy": acc
    })

# --- SVM ---
kernels = ["linear", "rbf"]
```

```
C_values = [0.1, 1, 2, 5, 10]
gamma_values = ["scale", "auto"]
for k, c, g in itertools.product(kernels, C_values, gamma_values):
    model = SVC(kernel=k, C=c, gamma=g, random_state=random_state)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    results.append({
        "Model": "SVM",
        "Params": f"kernel={k}, C={c}, gamma={g}",
        "Accuracy": acc
    })

# --- Таблица ---
df_results = pd.DataFrame(results)
df_results = df_results.sort_values(by=["Model", "Accuracy"], ascending=[True, False])
print(df_results)

# --- График ---
plt.figure(figsize=(12,6))
for model in df_results["Model"].unique():
    subset = df_results[df_results["Model"] == model]
    plt.plot(subset["Params"], subset["Accuracy"], marker="o", label=model)

plt.xticks(rotation=90)
plt.ylabel("Accuracy")
plt.title("Comparison of classifiers with different parameters (Test set)")
plt.legend()
plt.tight_layout()
plt.show()
```