ЛАБОРАТОРНАЯ РАБОТА №2.
НАИВНЫЙ БАЙЕСОВСКИЙ КЛАССИФИКАТОР. МЕТОД ОПОПРНЫХ ВЕКТОРОВ

(Продолжительность лабораторного занятия – 4 часа)

А. НАЗНАЧЕНИЕ И КРАТКАЯ ХАРАКТЕРИСТИКА РАБОТЫ

В процессе выполнения настоящей работы закрепляются знания студентов по разделам «Метрики качества классификации», «Наивный байесовский классификатор» и «Метод опорных векторов» курса «Применение методов искусственного интеллекта в электроэнергетике». Работа имеет экспериментальный характер и включает анализ данных и работы алгоритмов машинного обучения.

Целью работы является получение практических навыков работы с моделями байесовского классификатора и метода опорных векторов в программной среде Python.

Б. СОДЕРЖАНИЕ РАБОТЫ

Работа содержит:

1. Анализ, предварительную предобработку и визуализацию данных.
2. Обучение и применение наивного байесовского классификатора и модели опорных векторов с оптимальными параметрами.
3. Построение и визуализация матрицы ошибок и ROC-кривой для моделей с оптимальными выбранными параметрами, а также расчет метрики AUC-ROC.

Работа выполняется на компьютерах в интерактивной среде разработки JupyterLab.

В. ЗАДАНИЕ НА РАБОТУ В ЛАБОРАТОРИИ

1. Загрузить анализируемые данные, выданные преподавателем.
2. Построить круговую диаграмму для принимаемых значений целевой переменной.
3. Построить столбиковую диаграмму для двадцати наиболее часто встречающихся слов в обоих классах.
4. Выполнить токенизацию текстового признака, исключив неинформативные часто встречающиеся слова.
5. Найти оптимальный параметр сглаживания alpha для наивного байесовского классификатора по метрикам precision и accuracy.
6. Построить зависимость метрики accuracy на обучающих и тестовых данных от варьируемого параметра. Построить матрицы ошибок для модели с оптимальным выбранным параметром

7. Построить ROC-кривую и рассчитать метрику AUC-ROC.

8. Найти оптимальный параметр регуляризатора C для модели опорных векторов по метрикам precision и accuracy.

9. Повторить пункты 6 и 7 для модели опорных векторов.

Г. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К РАБОТЕ В ЛАБОРАТОРИИ

К пункту 1.

Для того, чтобы загрузить данные в формате «.csv» используйте метод read_csv библиотеки Pandas, аргументом которого является путь к файлу. Метод возвращает объект класса DataFrame.

К пункту 2.

Используйте метод plot класса DataFrame, чтобы построить круговую диаграмму целевых значений:

```
import pandas as pd

import matplotlib.pyplot as plt

target = pd.value_counts(data['target'])

target.plot(kind = 'pie')

plt.title('pie chart')

plt.ylabel('')
```

Где data['target'] столбец целевой переменной в объекте класса DataFrame.

К пункту 3.

Прежде необходимо вычислить наиболее часто встречающиеся слова в обоих классах. Для этого можно воспользоваться классом collections.Counter и методом most_common:

```
from collections import Counter

ham_words = Counter(" ".join(data[data['target']=='ham']['text']). \

split()).most_common(20)

df_ham_words = pd.DataFrame.from_dict(ham_words)

df_ham_words = df_ham_words.rename(columns={0: 'words in non-spam', 1:'count'})
```

Для построение столбиковой диаграммы можно воспользоваться методом plot.bar класса DataFrame:

```
import numpy as np

df_ham_words.plot.bar(legend = False)

y_pos = np.arange(len(df_ham_words['words in non-spam']))
```

```
plt.xticks(y_pos, df_ham_words['words in non-spam'])
plt.title('more frequent words in non-spam messages')
plt.xlabel('words')
plt.ylabel('number')
plt.show()
```

К пункту 4.

Токенизация – это преобразование текста в признаковый вектор, отражающий информационное содержание письма. В данной задаче для выполнения токенизации можно воспользоваться классом CountVectorizer библиотеки Scikit-learn:

```
from sklearn import feature_extraction
tokenizer = feature_extraction.text.CountVectorizer(stop_words = 'english')
X = tokenizer.fit_transform(data['text'])
```

К пункту 5.

Прежде, чем приступать к обучению классификатора, необходимо разбить выборку на обучающую и тестовую:

```
from sklearn import model_selection
data['target'] = data['target'].map({'spam':1, 'ham':0})
X_train, X_test, y_train, y_test = model_selection \
.train_test_split(X, data['target'], test_size = 0.33)
```

Модель наивного байесовского классификатора импортируйте из библиотеки Scikit-learn:

```
from sklearn.naive_bayes import MultinomialNB
```

Попробуйте найти оптимальный параметр alpha в диапазоне от 0,1 до 20 с шагом 0,1:

```
alpha_range = np.arange(0.1, 20, 0.1)
```

Метрики качества можно посчитать, воспользовавшись библиотекой Scikit-learn:

```
from sklearn import metrics
metrics.accuracy_score(y_test, y_predict)
metrics.recall_score(y_test, y_predict)
metrics.precision_score(y_test, y_predict)
```

Для поиска оптимального параметра по полученным данным удобно свести значения метрик в одну таблицу:

```
matrix = np.matrix(np.c_[alpha_range, train_score, test_score, test_recall, test_precision])

models = pd.DataFrame(data = matrix, columns = ['alpha', 'train accuracy', 'test accuracy', 'test recall', 'test precision'])

best_index = models['test precision'].idxmax()

best_index = models[models['test precision']==best_value]['test accuracy']. \
idxmax()
```

Где best_value – лучшее значение метрики precision.

По найденному оптимальному параметру обучите новый классификатор, чтобы использовать его при выполнении следующих пунктов:

```
model = MultinomialNB(alpha = alpha_range[best_index])

model.fit(X_train, y_train)
```

К пункту 6.

Построить зависимость двух векторов можно, воспользовавшись библиотекой Matplotlib.

Используйте библиотеки Scikit-learn и Pandas для построения матрицы ошибок:

```
from sklearn.metrics import confusion_matrix

confusion_matrix = confusion_matrix(y_test, model.predict(X_test))

pd.DataFrame(data = confusion_matrix, columns = ['predicted ham', 'predicted spam'], index = ['actual ham', 'actual spam'])
```

К пункту 7.

Рассчитать значения кривой ROC можно, воспользовавшись библиотекой Scikit-learn:

```
from sklearn.metrics import roc_curve

y_pred_pr = model.predict_proba(X_test)[:,1]

fpr, tpr, threshold = metrics. roc_curve (y_test, y_pred_pr)
```

Значение AUC-ROC можно посчитать, используя библиотеку Scikit-learn:

```
from sklearn.metrics import auc

roc_auc = metrics.auc(fpr, tpr)
```

Визуализировать кривую можно, использовав библиотеку Matplotlib:

```
plt.title('Receiver Operating Characteristic')

plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)

plt.legend(loc = 'lower right')
```

```
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.grid('on')
plt.show()
```

К пункту 8.

Загрузите модель опорных векторов из библиотеки Scikit-learn:

```
from sklearn.svm import SVC
```

Оптимальное значение параметра C попробуйте найти в диапазоне от 0.01 до 3 с шагом 0.1.

Д. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ОФОРМЛЕНИЮ ИСПОЛНИТЕЛЬНОГО ОТЧЕТА

Исполнительный отчет должен включать в себя:

- титульный лист с названием лабораторной работы и фамилией студента;
- цель лабораторной работы;
- листинг кода;
- результаты работы каждого пункта задания в виде графиков Matplotlib с подписанными осями;
- выводы о проделанной работе.

Вопросы к лабораторной работе №2

1. Какие проблемы имеет метрика качества классификации accuracy?
2. Чем отличаются метрики precision и recall от accuracy?
3. К каким алгоритмам классификации можно применять метрику AUC-PRC?
4. Как построить кривую precision-recall?
5. По каким осям строится ROC кривая?
6. В каких случая лучше применять AUC-PRC? А в каких – AUC-ROC?

7. Какого типа задачи наивный байесовский классификатор решает лучше метрических и линейных методов?
8. Чем отличаются оптимальный и наивный байесовские классификаторы?
9. В чем различие параметрических и непараметрических моделей восстановления плотностей?
10. Что называют опорными векторами в модели опорных векторов?
11. Какой принцип построения разделяющей гиперплоскости заложен в модели опорных векторов?