НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій (ІСТ)

ЗВІТ ДО ЛАБОРАТОРНОЇ РОБОТИ №6

з дисципліни

«Прикладні задачі машинного навчання»

Тема: «Прикладна задача машинного навчання»

Перевірив:

Андрій Олександрович Нестерук

Виконали:

Студенти групи ІК-33

Майстренко Тимофій Валентинович

Вересоцький Арсеній Юрійович

2025

Лабораторна робота №6

Тема: Прикладна задача машинного навчання:

Завдання

1. **Створити, навчити і апробувати багатошарову нейронну мережу з прямою передачею сигналу для ухвалення рішення про зарахування до Університету абітурієнтів, які здали вступні іспити з математики, англійської та української мови.**

Правила прийому наступні:

1. Рейтинг абітурієнтів формується за формулою 0,4 БМ+0,3БА+0,3БУ, де БМ-бал з іспиту з математики, БА-бал з іспиту з англійської мови, БУ-бал з іспиту з української мови.
2. Мінімальний прохідний бал на вступ 160 для абітурієнтів без пільг.
3. З математики для абітурієнтів без пільг мінімальний бал іспиту не може бути менший 140 балів.
4. Абітурієнти, які мають пільги, зараховуються при мінімумі 120 балів з усіх іспитів і їх рейтинг не може бути меншим ніж 144 бали
5. Університет може прийняти на навчання 350 абітурієнтів, з них не більше 10% це абітурієнти з пільгами.
6. Статистика минулих років показує, що в середньому до Університету подають документи 1500 абітурієнтів.

Рейтинг абітурієнтів формується за формулою:

$$0.4*M+0.3*A+0.3*У$$

М – бал з математики А – бал з англійської мови У – бал з української мови

Абітурієнтів повинно бути ~1500.

Може бути прийнято максимум 350 абітурієнтів, з яких лише 10% можуть бути пільговиками.

Імпортуємо бібліотеки для роботи

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, f1_score
from tensorflow.keras.regularizers import l2
from tensorflow.keras.optimizers import Adam, SGD
```

Та прописуємо загальні константи які ми будемо використовувати для роботи

```
SIZE = 15000
LOC_PRIVILEGED = 155
LOC_REGULAR = 175
PRIVILEGE_RATIO = 0.05
MIN_SCORE = 100
MAX_SCORE = 200
N_PRIV = int(SIZE * PRIVILEGE_RATIO)
N_NON_PRIV = SIZE - N_PRIV
SCALE_PRIV = 12
SCALE_REG = 15
```

SIZE - Загальна кількість вступників

LOC_PRIVILEGED - Середній бал для студентів пільговиків

LOC_REGULAR - Середній бал для студентів без пільг

PRIVILEGE_RATIO - Відсоток студентів пільговиків від загальної кількості

MIN_SCORE - Мінімальний бал за предмет

MAX_SCORE - Максимальний бал за предмет

N_PRIV - Кількість студентів без пільг

N_NON_PRIV - Кількість студентів пільговиків

SCALE_PRIV - Стандартне відхилення (σ) нормального розподілу середнього балу для студентів без пільг

SCALE_REG - Стандартне відхилення (σ) нормального розподілу середнього балу для студентів пільговиків

Створюємо функцію для генерації 15 000 студентів, серед яких 5 % мають пільги, задаючи для кожної групи власні середні та стандартні відхилення. Функція генерує вибірки з нормального розподілу з заданими loc та scale, відкидає значення за межами [100, 200] і продовжує відбір, доки не назбирає потрібну кількість. Далі для кожного з трьох предметів — математики, англійської та української — формуються масиви розміром N_NON_PRIV (звичайні) плюс N_PRIV (привілейовані), а вектор privileges містить 0 для перших N_NON_PRIV елементів і 1 для наступних N_PRIV.

```
# Генерація балів (нормальний розподіл, обрізаний до [100, 200])
def generate_scores(loc, scale, size, low, high):
  result = []
  while len(result) < size:
      sample = np.random.normal(loc, scale, size)
      filtered = sample[(sample >= low) & (sample <= high)]
      result.extend(filtered)
  return np.array(result[:size])

math = np.concatenate([
    generate_scores(LOC_REGULAR, SCALE_PRIV, size=N_NON_PRIV, low=MIN_SCORE, high=MAX_SCORE),
    generate_scores(LOC_PRIVILEGED, SCALE_REG, size=N_PRIV, low=MIN_SCORE, high=MAX_SCORE)
])

eng = np.concatenate([
    generate_scores(LOC_REGULAR, SCALE_PRIV, size=N_NON_PRIV, low=MIN_SCORE, high=MAX_SCORE),
    generate_scores(LOC_PRIVILEGED, SCALE_REG, size=N_PRIV, low=MIN_SCORE, high=MAX_SCORE)
])

ukr = np.concatenate([
    generate_scores(LOC_REGULAR, SCALE_PRIV, size=N_NON_PRIV, low=MIN_SCORE, high=MAX_SCORE),
    generate_scores(LOC_PRIVILEGED, SCALE_REG, size=N_PRIV, low=MIN_SCORE, high=MAX_SCORE)
])

privileges = np.array([0] * N_NON_PRIV + [1] * N_PRIV)
```

Після генерації даних обраховуємо рейтинг за формулою

$$0.4*М+0.3*А+0.3*У$$

М – бал з математики А – бал з англійської мови У – бал з української мови, та створюємо датафрейм з усіма даними.

```
rating = 0.4 * math + 0.3 * eng + 0.3 * ukr
```

```python
# Створення датафрейму
df = pd.DataFrame({
    'math': math,
    'eng': eng,
    'ukr': ukr,
    'privilege': privileges,
    'rating': rating,
}).sort_values(by='rating', ascending=False)
```

До нашого датафрейму необхідно додати колонку чи зарахували студента, тому прописуємо константи мінімальних вимог, та створюємо функцію яка перевіряє студента на мінімальні вимоги.

```python
# Мінімальні вимоги
MIN_PASS_RATING_NON_PRIV = 160
MIN_MATH_NON_PRIV = 140
MIN_ENG_NON_PRIV = 120
MIN_UKR_NON_PRIV = 120

MIN_PASS_RATING_PRIV = 144
MIN_EACH_PRIV = 120
```

MIN_PASS_RATING_NON_PRIV - Мінімальний рейтинг для студентів без пільг

MIN_MATH_NON_PRIV - Мінімальний бал за математику для студентів без пільг

MIN_ENG_NON_PRIV - Мінімальний бал за англійську мову для студентів без пільг

MIN_UKR_NON_PRIV - Мінімальний бал за українську мову для студентів без пільг

MIN_PASS_RATING_PRIV - Мінімальний рейтинг для студентів пільговиків

MIN_EACH_PRIV - Мінімальний бал за предмети для студентів пільговиків

Напишемо код який буде обирати із датафрейму студентів які попадають під вимоги, окремо з пільгами та без. Потім ставимо обмеження на максимальну кількість зарахованих через відношення `SIZE*350/1500`. Далі об'єднуємо зарахованих студентів, решту у колонці `accepted` ставимо 0, тобто не зараховано.

```
# Відбір тих, хто відповідає вимогам
eligible_non_priv = df[
    (df["privilege"] == 0) &
    (df["math"] >= MIN_MATH_NON_PRIV) &
    (df["eng"] >= MIN_ENG_NON_PRIV) &
    (df["ukr"] >= MIN_UKR_NON_PRIV) &
    (df["rating"] >= MIN_PASS_RATING_NON_PRIV)
].sort_values(by="rating", ascending=False)

eligible_priv = df[
    (df["privilege"] == 1) &
    (df[["math", "eng", "ukr"]].min(axis=1) >= MIN_EACH_PRIV) &
    (df["rating"] >= MIN_PASS_RATING_PRIV)
].sort_values(by="rating", ascending=False)

# Відбір до проходження
max_accepted = int(SIZE*350/1500)
accepted_priv = eligible_priv.head(int(max_accepted*0.1))
remaining_slots = max_accepted - len(accepted_priv)
accepted_non_priv = eligible_non_priv.head(remaining_slots)

# Об'єднання зарахованих
final_accepted = pd.concat([accepted_non_priv, accepted_priv])
final_accepted["accepted"] = 1

# Додаємо статус до всього dataframe
df["accepted"] = 0
df.loc[final_accepted.index, "accepted"] = 1
```

Виведемо частину списку

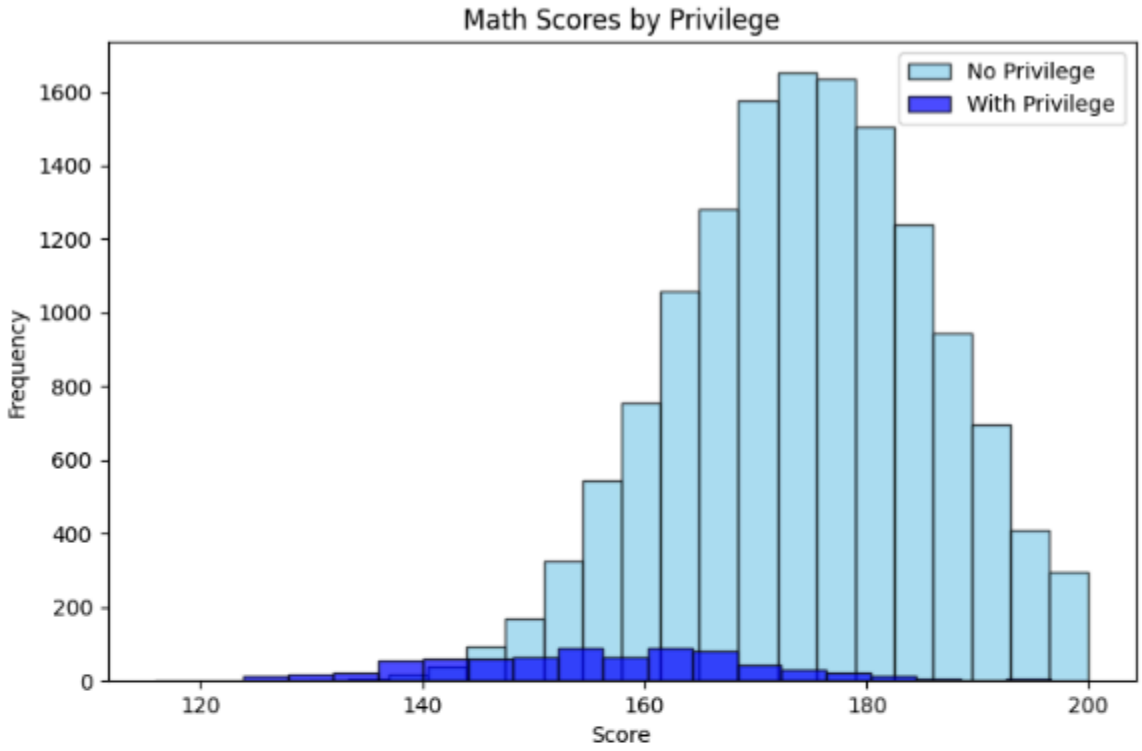
```
print(df.head())
```

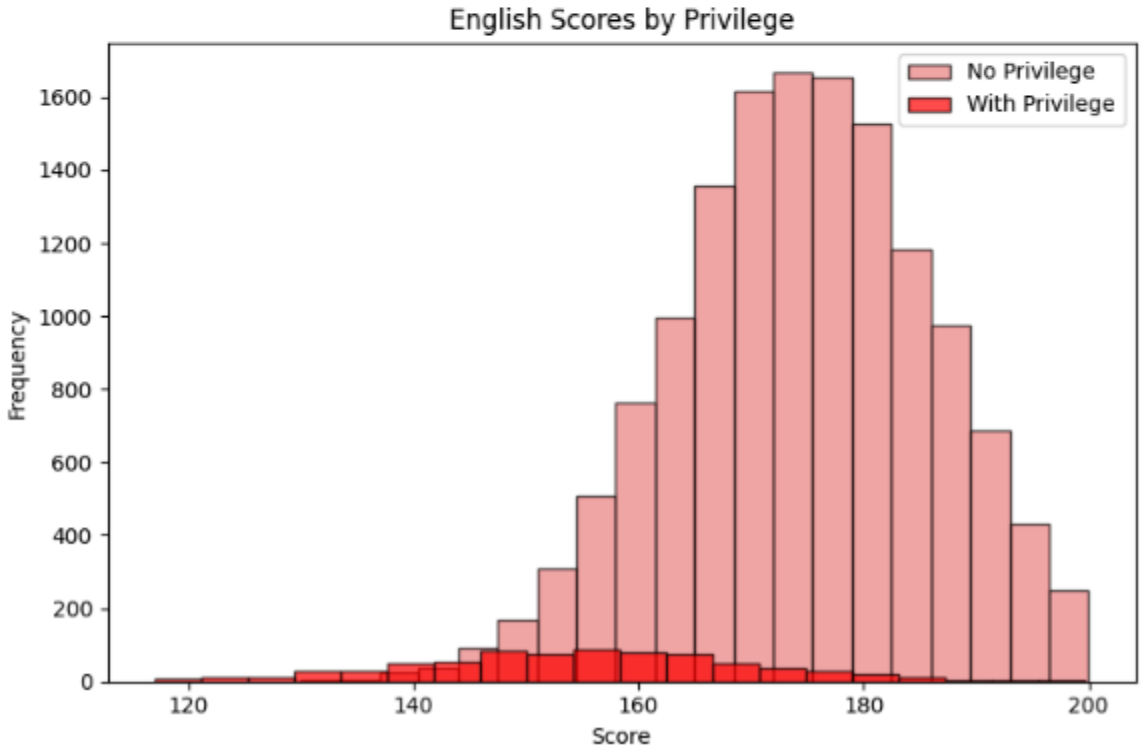
```
            math         eng         ukr  privilege      rating  accepted
9455   191.404575  198.381954  196.523735          0  195.033537         1
8896   198.172197  187.471355  196.223716          0  194.377400         1
14222  195.551465  196.956691  190.208321          0  194.370090         1
2695   197.721997  192.178050  190.740618          0  193.964399         1
952    199.172148  186.357690  194.436519          0  193.907122         1
```

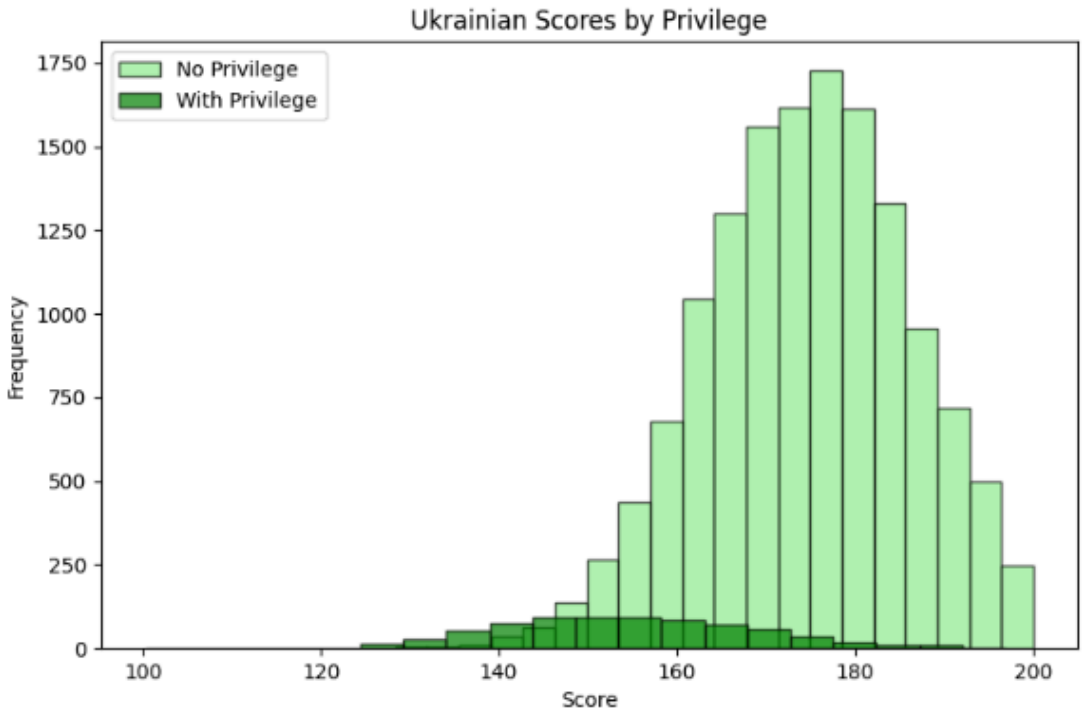
Для кращого розуміння даних, які ми згенерували, напишемо код який покаже розподіл оцінок за три предмети та рейтинг студентів з пільгами та без.

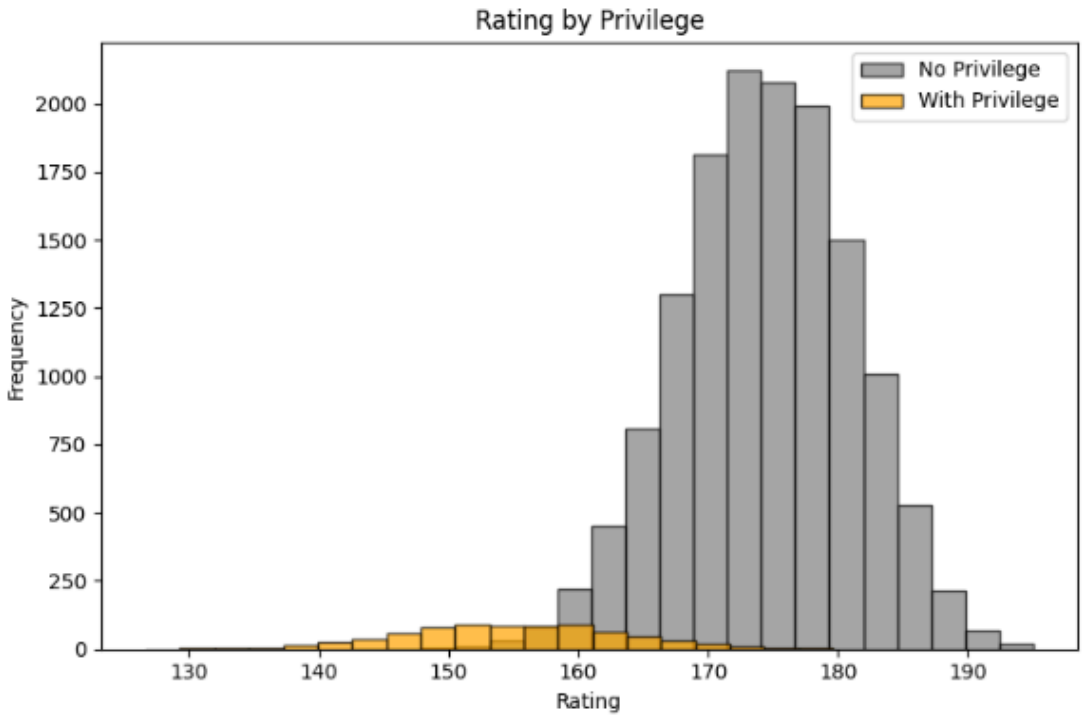
```
df_priv = df[df["privilege"] == 1]     # з пільгами
df_non_priv = df[df["privilege"] == 0] # без пільг

plt.figure(figsize=(15, 10))

# Математика
plt.subplot(2, 2, 1)
plt.hist(df_non_priv["math"], bins=20, alpha=0.7, label="No Privilege", color="skyblue", edgecolor="black")
plt.hist(df_priv["math"], bins=20, alpha=0.7, label="With Privilege", color="blue", edgecolor="black")
plt.title("Math Scores by Privilege")
plt.xlabel("Score")
plt.ylabel("Frequency")
plt.legend()

# Українська
plt.subplot(2, 2, 2)
plt.hist(df_non_priv["ukr"], bins=20, alpha=0.7, label="No Privilege", color="lightgreen", edgecolor="black")
plt.hist(df_priv["ukr"], bins=20, alpha=0.7, label="With Privilege", color="green", edgecolor="black")
plt.title("Ukrainian Scores by Privilege")
plt.xlabel("Score")
plt.ylabel("Frequency")
plt.legend()

# Англійська
plt.subplot(2, 2, 3)
plt.hist(df_non_priv["eng"], bins=20, alpha=0.7, label="No Privilege", color="lightcoral", edgecolor="black")
plt.hist(df_priv["eng"], bins=20, alpha=0.7, label="With Privilege", color="red", edgecolor="black")
plt.title("English Scores by Privilege")
plt.xlabel("Score")
plt.ylabel("Frequency")
plt.legend()

# Рейтинг
plt.subplot(2, 2, 4)
plt.hist(df_non_priv["rating"], bins=20, alpha=0.7, label="No Privilege", color="gray", edgecolor="black")
plt.hist(df_priv["rating"], bins=20, alpha=0.7, label="With Privilege", color="orange", edgecolor="black")
plt.title("Rating by Privilege")
plt.xlabel("Rating")
plt.ylabel("Frequency")
plt.legend()

plt.tight_layout()
plt.show()
```

Гістограма розподілу









Бачимо розподіл який відповідає нашим початковим параметрам, та розподіл який буде гарним для навчання моделі. Також бачимо одну таку примітивність, як студенти пільговики мають більший розподіл, та нижчий середній бал.

Виведемо чисельну характеристику даних

```
# Загальні межі балів
print("Загальні результати:")
print("Максимальний рейтинг:", df["rating"].max())
print("Мінімальний рейтинг:", df["rating"].min())
print()

# Зараховані непільговики
non_priv_accepted = df[(df["accepted"] == 1) & (df["privilege"] == 0)]
print("Непільговики (зараховані):")
print("Макс. рейтинг:", non_priv_accepted["rating"].max())
print("Мін. рейтинг:", non_priv_accepted["rating"].min())
print()

# Зараховані пільговики
priv_accepted = df[(df["accepted"] == 1) & (df["privilege"] == 1)]
print("Пільговики (зараховані):")
print("Макс. рейтинг:", priv_accepted["rating"].max())
print("Мін. рейтинг:", priv_accepted["rating"].min())
print()

# Незараховані непільговики
non_priv_rejected = df[(df["accepted"] == 0) & (df["privilege"] == 0)]
print("Непільговики (НЕ зараховані):")
print("Макс. рейтинг:", non_priv_rejected["rating"].max())
print("Мін. рейтинг:", non_priv_rejected["rating"].min())
print()

# Незараховані пільговики
priv_rejected = df[(df["accepted"] == 0) & (df["privilege"] == 1)]
print("Пільговики (НЕ зараховані):")
print("Макс. рейтинг:", priv_rejected["rating"].max())
print("Мін. рейтинг:", priv_rejected["rating"].min())
```

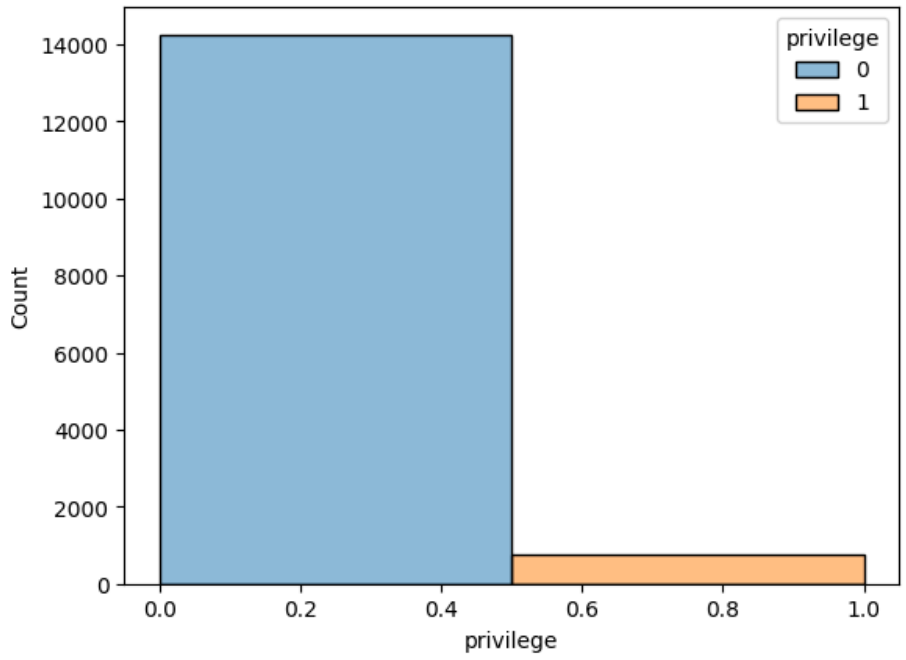
```
Загальні результати:

Максимальний рейтинг: 195.0335366502822

Мінімальний рейтинг: 126.67365092154684
```

```
Непільговики (зараховані):
Макс. рейтинг: 195.0335366502822
Мін. рейтинг: 179.61688999484534

Пільговики (зараховані):
Макс. рейтинг: 179.64096600539796
Мін. рейтинг: 155.7995536392563

Непільговики (НЕ зараховані):
Макс. рейтинг: 179.61651954585113
Мін. рейтинг: 142.72214162843

Пільговики (НЕ зараховані):
Макс. рейтинг: 160.32002230290254
Мін. рейтинг: 126.67365092154684
```

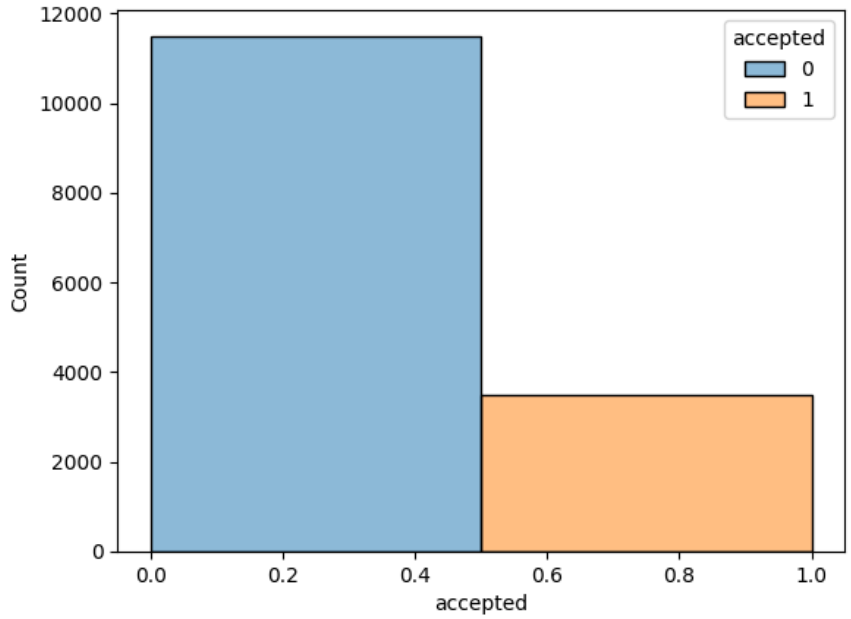
Згенеровані рейтинги лежать у діапазоні від приблизно 126,7 до 195,0. Непільговики, які пройшли відбір, мають значення в межах 179,6–195,0, пільговики – 155,8–179,6. Водночас серед відхилених непільговиків діапазон 142,7–179,6, а пільговиків – 126,7–160,3. Це вказує на те, що розподіл пільговиків зміщений у бік нижчих рейтингів порівняно з непільговиками.

Виведемо співвідношення абітурієнтів за наявністю пільги, а також співвідношення абітурієнтів за зарахуванням у заклад відповідно.

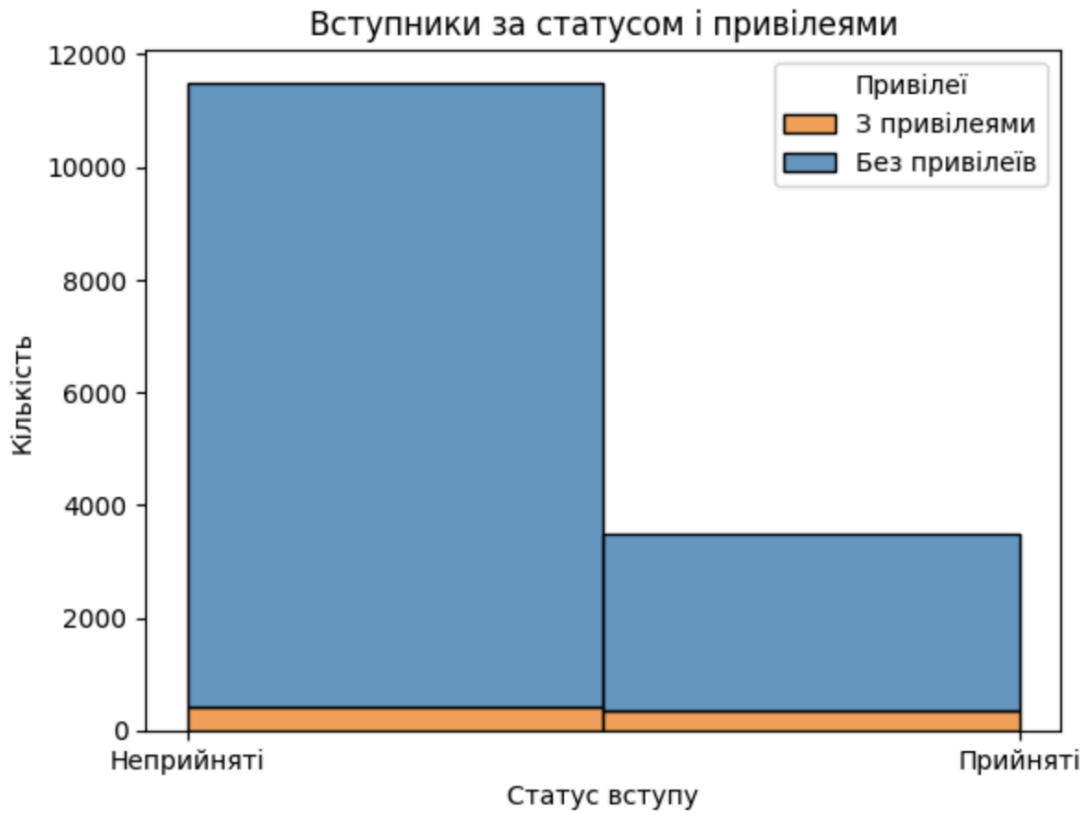
```
# Співвідношення абітурієнтів за наявністю пільги
sns.histplot(data=df, x='privilege', hue='privilege', bins=2)
plt.show()
```




```
# Співвідношення абітурієнтів за зарахуванням
sns.histplot(data=df, x='accepted', hue='accepted', bins=2)
plt.show()
```



```
sns.histplot(data=df, x='accepted', hue='privilege', multiple='stack', bins=2)
plt.xticks([0, 1], ['Неприйняті', 'Прийняті'])
plt.xlabel('Статус вступу')
plt.ylabel('Кількість')
plt.title('Вступники за статусом і привілеями')
plt.legend(title='Привілеї', labels=['З привілеями', 'Без привілеїв'])
plt.show()
```




```
Загальна кількість заяв: 15000
Прийнято: 3500
├── З привілеями: 350
└── Без привілеїв: 3150
Не прийнято: 11500
```

Далі формуємо тренувальний та тестовий набори, зберігаючи співвідношення вступників за пільгою

Спочатку з основного датафрейму відокремлюємо дві підвибірки: тих, кого було зараховано (accepted), і тих, кого не зарахували (not_accepted). Далі для зарахованих застосовуємо стратифіковане розділення на тренувальний і тестовий набори (10% для тесту). Для незарахованих використовується звичайне випадкове розділення, без стратифікації, оскільки серед цієї групи збереження співвідношення не є таким критичним. Після цього відбувається об’єднання тренувальних частин у єдиний тренувальний набір, і аналогічно — тестових частин у тестовий. Обидва набори перемішуються для уникнення впливу порядку записів на результати навчання.

На наступному етапі виділяються ознаки (math, eng, ukr, privilege) і цільова змінна (accepted) для тренувального та тестового наборів. Потім дані масштабуються за допомогою стандартного нормалізатора StandardScaler, щоб привести всі ознаки до одного масштабу.

```python
# Отримаємо зарахованих
accepted = df[df['accepted'] == 1]
not_accepted = df[df['accepted'] == 0]

# Стратифіковане розділення лише для зарахованих (щоб зберегти частку пільговиків)
accepted_train, accepted_test = train_test_split(
    accepted,
    test_size=0.1,
    stratify=accepted['privilege'],
    random_state=52
)

# Звичайне перемішане розділення для незарахованих
not_accepted_train, not_accepted_test = train_test_split(
    not_accepted,
    test_size=0.1,
    random_state=52
)

# Об'єднуємо назад
train = pd.concat([accepted_train, not_accepted_train])
test = pd.concat([accepted_test, not_accepted_test])

# Перемішуємо
train = train.sample(frac=1, random_state=52).reset_index(drop=True)
test = test.sample(frac=1, random_state=52).reset_index(drop=True)

X_train = train[['math', 'eng', 'ukr', 'privilege']].values
y_train = train['accepted'].values

X_test = test[['math', 'eng', 'ukr', 'privilege']].values
y_test = test['accepted'].values
```

```python
# Масштабуємо дані
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Основна функція build_model приймає параметри: архітектуру arch, яка визначає кількість нейронів у шарах, булеві значення dropout (використовувати регуляризацію Dropout чи ні), l2_reg (використовувати L2-регуляризацію чи ні), а також назву оптимізатора ('adam' або 'sgd').

Модель будується послідовно (Sequential). Перший шар отримує 4 вхідні ознаки (math, eng, ukr, privilege) через об'єкт Input. Далі додається перший повнозв'язний (Dense) шар із заданою кількістю нейронів і функцією активації ReLU. Якщо l2_reg=True, до цього шару застосовується L2-регуляризація (штраф на великі ваги), що допомагає зменшити

перенавчання. Наступні шари створюються аналогічно — їх кількість і розміри задаються елементами списку arch. Якщо ввімкнено Dropout, після кожного шару додається Dropout(0.3), що випадково обнуляє 30% виходів, щоб модель не переобучалась.

```
def build_model(arch, dropout=False, l2_reg=False, optimizer='adam'):
    model = Sequential()

    reg = l2(0.001) if l2_reg else None

    model.add(Input((4,)))
    # Перший шар (вхідний)
    model.add(Dense(arch[0], activation='relu', kernel_regularizer=reg))

    for units in arch[1:]:
        model.add(Dense(units, activation='relu', kernel_regularizer=reg))
        if dropout:
            model.add(Dropout(0.3))

    model.add(Dense(1, activation='sigmoid'))

    opt = Adam() if optimizer == 'adam' else SGD(learning_rate=0.01)

    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Вихідний шар — один нейрон з функцією активації sigmoid, яка видає значення від 0 до 1, що інтерпретується як ймовірність належності до класу "зарахований". Функція втрат — binary_crossentropy, стандартна для бінарної класифікації.

Після визначення моделі вона тренується на тренувальних даних (X_train, y_train) протягом 10 епох з розміром пакета 32. Також використовується 20% даних для валідації (validation_split=0.2). Далі модель передбачає значення для тестової вибірки (X_test). Значення округлюються до 0 або 1 на основі порогу 0.5. Оцінка виконується за метриками точності (accuracy_score) та F1-міри (f1_score), які потім зберігаються в список results разом із параметрами конфігурації.

Словник architectures містить 6 різних конфігурацій моделі — від простої (один шар із 8 нейронами) до складної з кількома шарами і

регуляризаціями. Кожна з них перевіряється з двома оптимізаторами, що дозволяє порівняти, як архітектура та метод оптимізації впливають на якість класифікації.

```
results = []

architectures = {
    "A": ([8], False, False),
    "B": ([8, 4], False, False),
    "C": ([16, 8, 4], False, False),
    "D": ([8, 4], True, False),
    "E": ([8, 4], False, True),
    "F": ([8, 4], True, True)
}

for name, (arch, dropout, l2reg) in architectures.items():
    for opt in ['adam', 'sgd']:
        model = build_model(arch, dropout, l2reg, optimizer=opt)
        history = model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_split=0.2)
        y_pred = (model.predict(X_test) > 0.5).astype(int)
        acc = accuracy_score(y_test, y_pred)
        f1 = f1_score(y_test, y_pred)
        results.append({
            "Model": name,
            "Arch": str(arch),
            "Dropout": dropout,
            "L2": l2reg,
            "Optimizer": opt,
            "Accuracy": acc,
            "F1": f1
        })
```

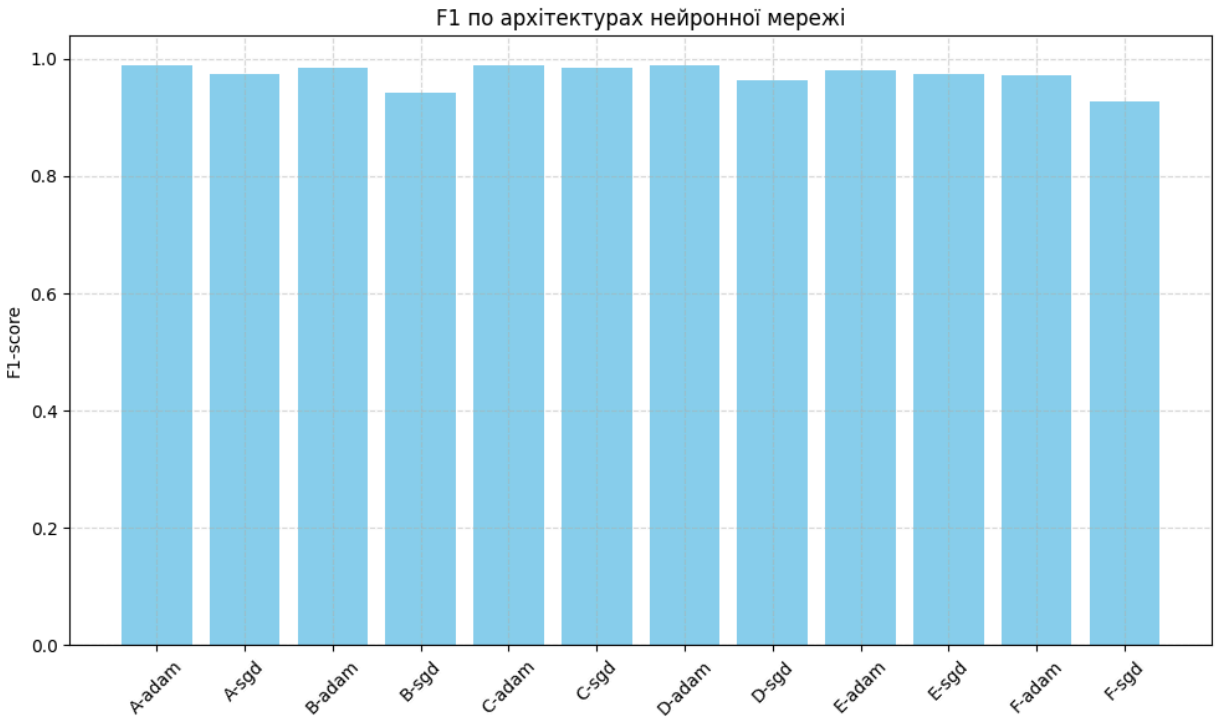
Після навчання різних моделей з різними архітектурами проаналізуємо результати.

```
# Аналіз результатів
results_df = pd.DataFrame(results)
print(results_df.sort_values(by='F1', ascending=False))
```

	Model	Arch	Dropout	L2	Optimizer	Accuracy	F1
0	A	[8]	False	False	adam	0.995333	0.989957
6	D	[8, 4]	True	False	adam	0.995333	0.989957
4	C	[16, 8, 4]	False	False	adam	0.994667	0.988439
5	C	[16, 8, 4]	False	False	sgd	0.993333	0.985632
2	B	[8, 4]	False	False	adam	0.993333	0.985507
8	E	[8, 4]	False	True	adam	0.991333	0.981077
9	E	[8, 4]	False	True	sgd	0.988667	0.975255
1	A	[8]	False	False	sgd	0.988000	0.973761
10	F	[8, 4]	True	True	adam	0.987333	0.972100
7	D	[8, 4]	True	False	sgd	0.984000	0.964497
3	B	[8, 4]	False	False	sgd	0.974000	0.942563
11	F	[8, 4]	True	True	sgd	0.968000	0.926829

Тут можна побачити, яка модель з якими налаштуваннями показала найкращі результати. Побудуємо додатково діаграму.

```
# Побудова діаграми F1
plt.figure(figsize=(10, 6))
plt.bar(results_df['Model'] + '-' + results_df['Optimizer'], results_df['F1'], color='skyblue')
plt.xticks(rotation=45)
plt.ylabel("F1-score")
plt.title("F1 по архітектурах нейронної мережі")
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.savefig("f1_by_model.png")
plt.show()
```



Хоча можна самому побачити яка модель є найкращою, проте напишемо код, який знайде найкращу модель порівнюючи параметр F1.

```
# Знайти найкращу модель за F1
best_model_info = results_df.sort_values(by='F1', ascending=False).iloc[0]
best_model_name = best_model_info['Model']
print("Найкраща модель:", best_model_info)
```

```
Найкраща модель: Model              A
Arch                [8]
Dropout           False
L2                False
Optimizer          adam
Accuracy       0.995333
F1             0.989957
Name: 0, dtype: object
```

Також отримаємо змінні найкращої моделі для побудови та тренування

```
# Створити її заново
best_arch = eval(best_model_info['Arch'])
best_dropout = best_model_info['Dropout']
best_l2 = best_model_info['L2']
best_opt = best_model_info['Optimizer']
```

```
# Побудова та тренування найкращої моделі
best_model = build_model(best_arch, dropout=best_dropout, l2_reg=best_l2, optimizer=best_opt)
history = best_model.fit(X_train, y_train, epochs=30, batch_size=32, validation_split=0.2)
```

```
Epoch 1/30
338/338 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7918 - loss: 0.5722 - val_accuracy: 0.9352 - val_loss: 0.3708
Epoch 2/30
338/338 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9507 - loss: 0.3174 - val_accuracy: 0.9678 - val_loss: 0.1980
```

...

```
Epoch 29/30
338/338 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9974 - loss: 0.0202 - val_accuracy: 0.9985 - val_loss: 0.0211
Epoch 30/30
338/338 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9971 - loss: 0.0209 - val_accuracy: 0.9956 - val_loss: 0.0214
```

Збережемо модель та перевіримо точність

```
# Збереження
best_model.save('best_admission_model.keras')
print("Модель збережена у файлі best_admission_model.keras")
```

```
Модель збережена у файлі best_admission_model.keras
```

```
# Точність
loss, accuracy = best_model.evaluate(X_test, y_test)
print(f"Точність на тестовій вибірці: {accuracy:.4f}")
```

```
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9935 - loss: 0.0227
Точність на тестовій вибірці: 0.9960
```

Точність складає 99.6%, що є високим показником, проте глянемо класифікаційний звіт, матрицю похибок, графік точності.

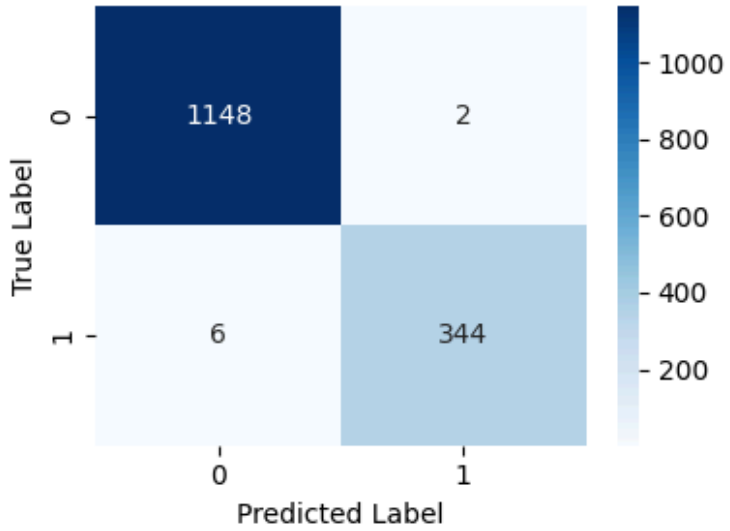
За допомогою classification_report обчислюються основні метрики: точність (accuracy), точність позитивного класу (precision), повнота (recall) та F1-міра (F1-score). Отримані результати показують, що модель класифікує з дуже високою точністю (Accuracy ≈ 0.9947), майже не помиляється в позитивних передбаченнях (Precision ≈ 0.9945) і з високою повнотою виявляє прийнятих абітурієнтів (Recall ≈ 0.9906). F1-міра як середнє між precision і recall — 0.9925 — свідчить про стабільну роботу моделі в обох класах.

```python
# Повний звіт
y_pred = (best_model.predict(X_test) > 0.5).astype("int32")
report = classification_report(y_test, y_pred, output_dict=True)

print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {report['macro avg']['precision']:.4f}")
print(f"Recall: {report['macro avg']['recall']:.4f}")
print(f"F1-score: {report['macro avg']['f1-score']:.4f}")
```

```
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
Accuracy: 0.9947
Precision: 0.9945
Recall: 0.9906
F1-score: 0.9925
```

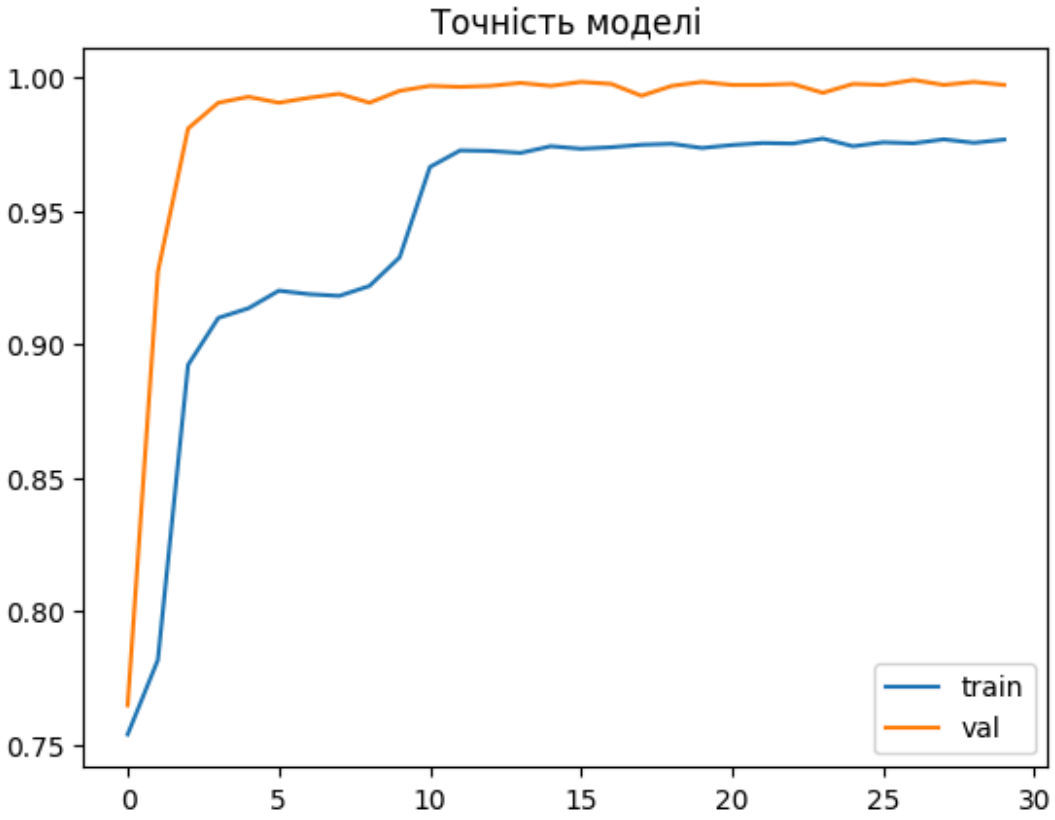
Далі візуалізація матриці помилок. Вона чітко показує, що з 1150 неприйнятих абітурієнтів лише 2 були помилково класифіковані як прийняті, а з 350 прийнятих — лише 6 помилково визнані неприйнятими.

```python
# Матриця помилок
plt.figure(figsize=(4, 3))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()
```



Графік ілюструє точність моделі протягом епох тренування як на тренувальній, так і на валідаційній вибірці. Видно, що вже після перших 5 епох модель досягає точності понад 95%, а надалі плавно покращує її до близько 99%

```
# Графік точності
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='val')
plt.legend()
plt.title("Точність моделі")
plt.show()
```



Перевіримо як модель реагує на аномальну дані

```
# Приклад одного "аномального" абітурієнта
sample = np.array([[200, 200, 100, 0]])  # Без пільг
sample_scaled = scaler.transform(sample)  # масштабування, як для train-даних

prediction = best_model.predict(sample_scaled)
predicted_label = (prediction > 0.5).astype(int)

print(f"Ймовірність зарахування: {prediction[0][0]:.4f}")
print(f"Клас: {'Зараховано' if predicted_label[0][0] == 1 else 'Не зараховано'}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
Ймовірність зарахування: 0.0000
Клас: Не зараховано
```

Вхідний вектор містить оцінки 200 з математики, 200 з англійської, 100 з української мови, а абітурієнт не має пільг.

Проводимо візуалізацію результатів моделі на тестовій вибірці. Спочатку модель генерує ймовірності зарахування для кожного вступника, а потім за порогом 0.5 формуються бінарні передбачення: 1 — "прийнятий", 0 — "неприйнятий".

Як бачимо, наша нейронна мережа вважає, що ймовірність зарахування такого абітурієнта дуже мала. Щоб зрозуміти це, вона не обраховувала рейтинг студента. Такого висновку вона дійшла через те, що її параметри(ваги) були навчені на даних, де подібні абітурієнти не проходили відбір через те, що не набрали достатню кількість балів з предмету.

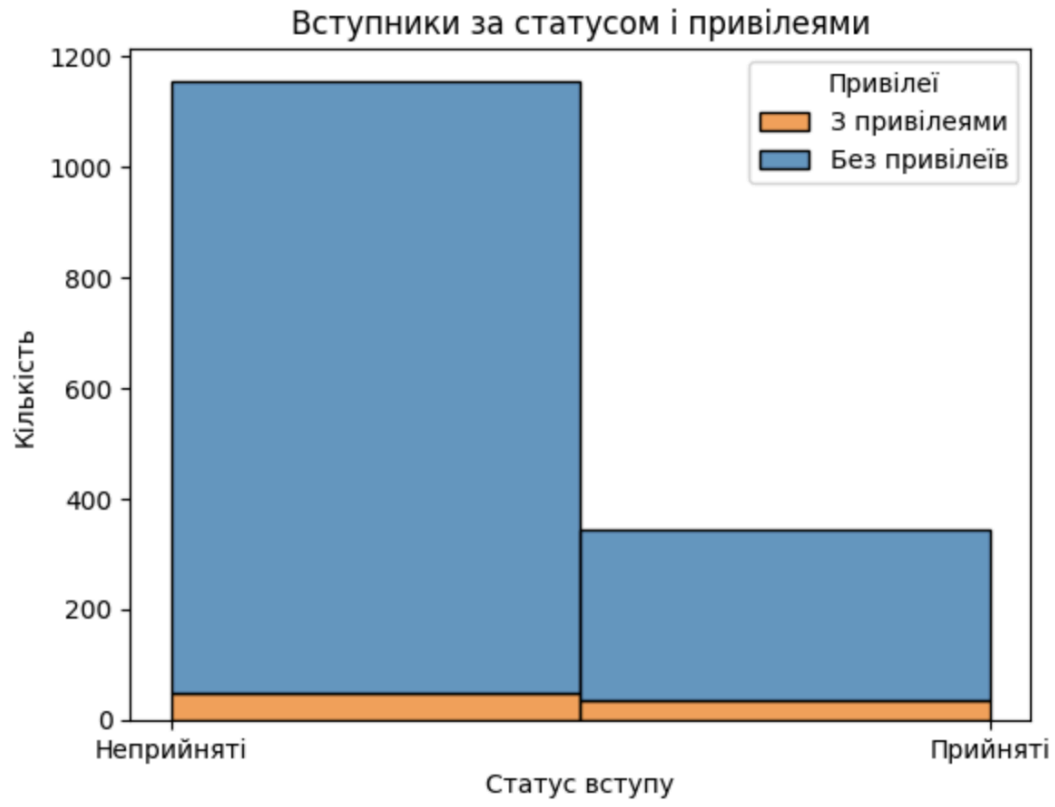
Після чого будується гістограма за аналогією до попередньої, але вже на основі **передбачених**, а не фактичних даних.

Графік показує, що більшість неприйнятих — це студенти без пільг, однак пільговики також присутні в обох групах. Модель добре дотримується співвідношення, оскільки пільговиків серед прийнятих видно небагато, що відповідає правилу 10% (яке було закладене в логіку).

```
test['prediction_prob'] = best_model.predict(X_test)
test['predicted'] = (test['prediction_prob'] > 0.5).astype(int)
```

```
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 955us/step
```

```
sns.histplot(data=test, x='predicted', hue='privilege', multiple='stack', bins=2)
plt.xticks([0, 1], ['Неприйняті', 'Прийняті'])
plt.xlabel('Статус вступу')
plt.ylabel('Кількість')
plt.title('Вступники за статусом і привілеями')
plt.legend(title='Привілеї', labels=['З привілеями', 'Без привілеїв'])
plt.show()
```



Виведемо загальну інформацію

```
total_predicted_accepted = test['predicted'].sum()
total_predicted_rejected = len(test) - total_predicted_accepted
predicted_accepted_with_priv = test[(test['predicted'] == 1) & (test['privilege'] == 1)].shape[0]
predicted_accepted_without_priv = test[(test['predicted'] == 1) & (test['privilege'] == 0)].shape[0]

print(f"Загальна кількість заяв: {len(test)}")
print(f"Прийнято: {total_predicted_accepted}")
print(f"├── З привілеями: {predicted_accepted_with_priv}")
print(f"└── Без привілеїв: {predicted_accepted_without_priv}")
print(f"Не прийнято: {total_predicted_rejected}")
```

```
Загальна кількість заяв: 1500
Прийнято: 346
├── З привілеями: 36
└── Без привілеїв: 310
Не прийнято: 1154
```

Виведемо деякі рядки з датафрейму

	math	eng	ukr	privilege	rating	accepted	prediction_prob	predicted
554	195.505462	194.432293	191.101534	0	193.862333	1	1.000000e+00	1
1347	197.156210	186.428466	194.009074	0	192.993746	1	1.000000e+00	1
5	187.005137	193.665405	199.092975	0	192.629569	1	1.000000e+00	1
859	196.920821	179.964110	196.805729	0	191.799280	1	1.000000e+00	1
422	198.957001	173.753001	195.580442	0	190.382833	1	1.000000e+00	1
...	...	...	...	...	...	...	...	...
21	138.232688	132.747726	148.432916	1	139.647268	0	1.363153e-20	0
383	141.071079	137.477822	137.770650	1	139.002973	0	4.605350e-21	0
382	139.506774	135.615607	134.330043	1	136.786404	0	4.429170e-23	0
285	128.814173	148.669390	128.021650	1	134.532981	0	5.867554e-26	0
249	127.657755	165.221615	100.316361	1	130.724495	0	5.351984e-30	0

Та збережемо все в ексель

```
# Збереження в Excel
df_final[['math', 'eng', 'ukr', 'privilege', 'rating', 'prediction_prob']].to_excel("accepted_applicants.xlsx", index=False)
```

✓ 0.1s

Висновок

У межах лабораторної роботи було успішно реалізовано повний цикл розв’язання прикладної задачі машинного навчання: від генерації штучних даних про вступників до побудови, навчання та оцінювання багатошарової нейронної мережі для автоматичного прийняття рішення щодо зарахування абітурієнтів.

Було враховано умови відбору — окремо для абітурієнтів з пільгами та без них — відповідно до реальних вступних правил. Дані були згенеровані з урахуванням статистичного розподілу оцінок, а рейтинг обчислювався згідно з формулою 0.4 * математика + 0.3 * англійська + 0.3 * українська.

Застосовано стратифікований підхід до формування тренувального і тестового наборів для збереження балансу пільговиків. Створено і протестовано декілька архітектур нейронних мереж з різними гіперпараметрами, включаючи Dropout, L2-регуляризацію та різні оптимізатори (Adam і SGD). Найкраща модель показала високу якість класифікації з точністю 99.5%, F1-мірою 0.9925 і дуже низьким рівнем помилок, що підтверджується як класифікаційним звітом, так і матрицею помилок.

Модель адекватно реагує як на звичайні, так і на граничні ("аномальні") випадки, та зберігає задані обмеження, зокрема не перевищує ліміт 10% пільговиків серед зарахованих. Усі результати було візуалізовано та експортовано до таблиці для подальшого аналізу.