**Звіт до комп’ютерного практикуму №1.**

**Моделі машинного навчання**

**ПІБ:** Кононов М. А.

**Група:** ЗПІ-зп01

**Мета роботи:** ознайомитись з принципами функціонування, створення, навчання та використання моделей машинного навчання.

***З*авдання:** для обраної задачі класифікації (або регресії) на основі типового датасету створити модель машинного навчання, навчити її на датасеті, перевірити результат на тестовій вибірці.

**Номер варіанту:** 8

**Завдання для варіанту:** датасет Titanic (класифікація), модель багатошаровий персептрон.

**Засоби виконання практикуму:** мова програмування Python, середовище Keras, бібліотеки numpy, pandas та matplotlib.

**Набір даних (датасет):** Titanic – широко доступний набір статистичних даних який містить інформацію про пасажирів затонулого корабля, зокрема про їх вік, стать, клас, проїзний білет, каюту та стан після катастрофи. Наша задача полягає в розробці програми для прогнозування, чи буде пасажир живим.

**Попередня обробка даних:** Для вирішення поставленої задачі нами було розроблено два рішення: з детальною попередньою обробкою даних, та із зовсім незначною попередньою обробкою. В Solution 1 ми видаляли менш впливові стовбці таблиці (білет та номер кабіни), узагальнювали поле Name до інформації лише про клас (проста людина або заможна), після чого видалили стовбці Name та PassengerId. Крім цього, назва статі отримала числове значення, пасажири були поділені на різні вікові групи та «тарифні зони», також було встановлено, чи подорожує пасажир один. В другому рішенні (Solution 2) ми лише обробили порожні значення таблиці.

**Модель машинного навчання:** коротка характеристика, структура, вхідні вихідні параметри, програмний код створення.

Багатошаровий персептрон складається з безлічі сенсорних елементів (вхідних вузлів або вузлів джерела), які утворюють вхідний шар; одного або декількох прихованих шарів обчислювальних нейронів і одного вихідного шару нейронів. Вхідний сигнал розповсюджується по мережі в прямому напрямку, від шару до шару.

Багатошарові персептрони успішно застосовуються для вирішення різноманітних складних задач. При цьому навчання з вчителем виконується за допомогою алгоритму зворотного розповсюдження помилки. Вони мають три відмітні ознаки:

1. Кожен нейрон мережі має нелінійну функцію активації. Найпопулярнішою формою функції, що задовольняє вимозі нелінійності, є сигмоїдальна, визначувана логістичною функцією:

**D:\Desktop\image001.png**

де є індуковане локальне поле (тобто зважена сума всіх синаптичних входів плюс порогове значення) нейрона і вихід нейрона.

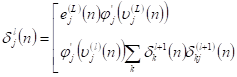
2. Мережа містить один або декілька шарів прихованих нейронів, що не є частиною входу або виходу мережі. Ці нейрони дозволяють мережі навчатися рішенню складних задач, послідовно витягуючи найважливіші ознаки з вхідного образу (вектора).

3. Мережа має високий степінь зв'язності, реалізований за допомогою синаптичних з'єднань.

Кожен нейрон в будь-якому шарі мережі пов'язаний зі всіма нейронами (вузлами) попереднього шару. На першому кроці алгоритм генерує синаптичну вагу і порогові значення за допомогою датчика рівномірно розподілених чисел з середнім значенням 0. На другому – пред'являються приклади навчання. У мережу подаються образи з навчальної множини (епохи). Для кожного образу послідовно виконуються прямий і зворотний проходи. Третій крок полягає у прямому проході. Приклад навчання представляється у вигляді пари, яка складається з вихідного вектора, який пред'являється вхідному шару сенсорних вузлів, та бажаного відгуку, що надається вихідному шару нейронів для формування сигналу помилки. Обчислюються індуковані локальні поля і функціональні сигнали мережі, проходячи по ній пошарово в прямому напрямку. Індуковане локальне поле нейрона шару обчислюється за формулою:



На четвертому кроці розраховуються локальні градієнти вузлів мережі наступним чином:



Зміна синаптичних ваг шару мережі виконується відповідно до узагальненого дельта-правила:



На останньому, п’ятому, кроці виконуються ітерації: послідовно здійснюються прямий і зворотний проходи, пред'являючи мережі всі приклади навчання з епохи, поки не буде досягнутий критерій зупинки.

Ми використовували два різних підходи для реалізацій даної моделі. Код першого з них:

input = Input(shape=(20,))

x = Dense(256,activation='relu')(input)

x = Dropout(0.8)(x)

x = Dense(256, activation='relu')(x)

x = Dense(128, activation='relu')(x)

x = BatchNormalization()(x)

x = Dense(128, activation='relu')(x)

x = Dense(64, activation='relu')(x)

x = Dense(64, activation='relu')(x)

output = Dense(1, activation='sigmoid')(x)

MLP = Model(input, output)

sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.6, nesterov=True)

adam = optimizers.Adam(lr=0.01, beta\_1=0.9, beta\_2=0.999, epsilon=None, decay=0.001, amsgrad=False)

MLP.compile(optimizer=adam, loss='binary\_crossentropy', metrics=['binary\_accuracy'])

split = int(X\_train.shape[0]\*0.9)

history = MLP.fit(X\_train.values, Y\_train.values, batch\_size=64, epochs=250,validation\_split=0.2, verbose=1)

predictions = MLP.predict(X\_test.values,verbose=0)

Другий спосіб реалізації (Solution 2):

def BuildModel():

model = Sequential()

model.add(Dense(7, input\_dim=7, activation='relu'))

model.add(Dense(2, activation='softmax'))

model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

model = BuildModel()

# підгонка моделі

history = model.fit(X\_transformed, labels, epochs=250, validation\_split=0.2, batch\_size=6, verbose=1)

pred = model.predict(X\_real\_test)

y\_pred\_test = pred.argmax(1)

pid = testdata[['PassengerId']].values

res = np.expand\_dims(y\_pred\_test,axis=1)

f = np.hstack((pid,res))

df = pd.DataFrame(f, columns = ['PassengerId', 'Survived'])

**Навчання моделі:** обґрунтування вибору алгоритму навчання, функції втрат, метрик тощо.

В обох рішеннях методи для побудови шарів персептрона дещо відрізняються: Solution 1 ґрунтується в першу чергу на декількох функціях relu (x = Dense(256, activation='relu')(x)), нормалізації пакетів (x = BatchNormalization()(x)), та x = Dropout(0.8)(x) (видалення зайвих сигналів).

У другому рішенні алгоритм побудови шарів виглядає простіше, базуючись лише на двох функціях, relu та softmax:

model.add(Dense(7, input\_dim=7, activation='relu'))

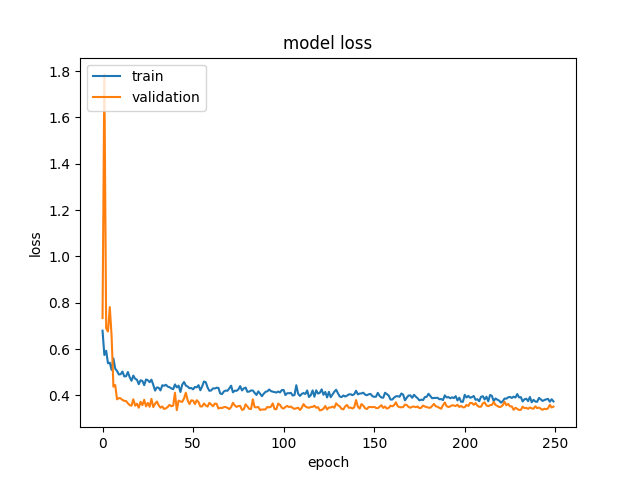
model.add(Dense(2, activation='softmax'))

Для навчання моделі в обох рішеннях ми застосовуємо метод fit, який у якості параметрів отримує наступні змінні:

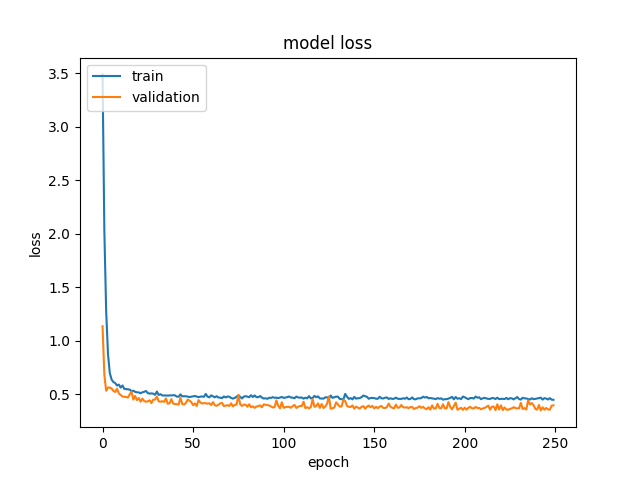
* X\_train.values – тренувальні дані без колонки Survived;
* Y\_train.values –дані, які містять лише колонку Survived (0 або 1);
* batch\_size – розмір пакету;
* epochs – кількість епох навчання;
* validation\_split – параметр валідації
* Verbose – ступінь інформатизації користувача (кількість інформації, яка виводиться на екран під час навчання).

В Solution 2 ці параметри виглядають так само, тільки самі дані для навчання та код їх відбору дещо відрізняється (виглядає простіше).

Для першого рішення ми отримали такий графік функції втрат:



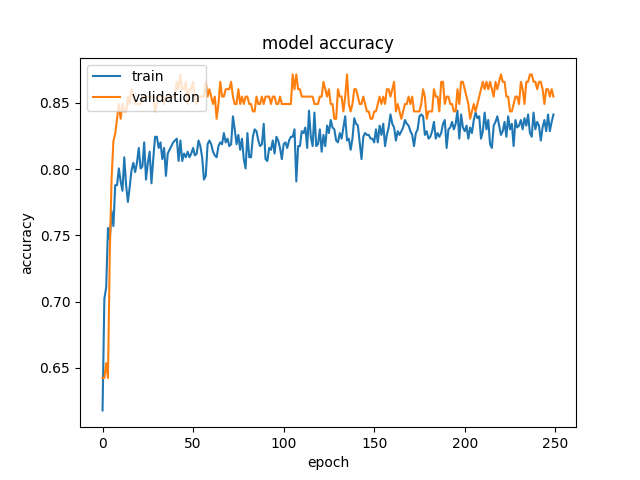
Для другого:



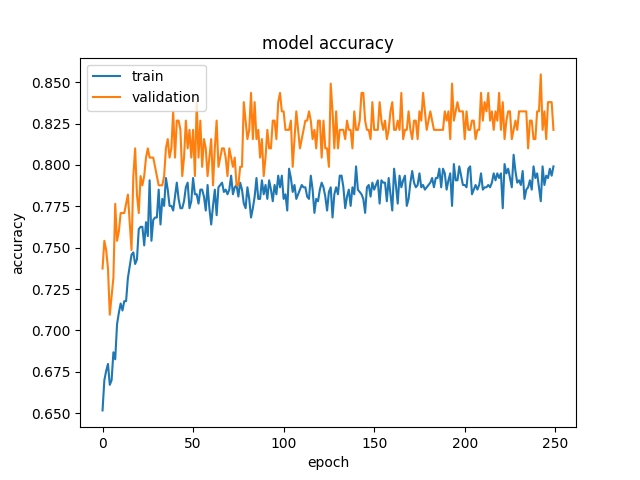
Як ми бачимо, вони виглядають майже повністю однаково. З них ми можемо зробити висновок, що кількість втрат перестає знижуватись після 50 епох, з незначними коливаннями залишаючись на майже однаковому рівні, що свідчить про можливість їх зменшення для отримання тих самих результатів.

**Результати навчання:** числові результати та динаміка навчання, бажано наочно представити результати.

Після прогонки в Solution 1 ми отримали таку точність:



Точність, яку дає Solution 2:



Ми бачимо, що використання двох майже повністю різних реалізацій багатошарового персептрона, а також суттєва попередня обробка навчальних даних у першому рішенні не призвели до збільшення точності роботи моделі. При роботі з навчальними даними вона виявилась дещо меншою, ніж під час валідації реальної моделі, що вважається закономірним.

**Оцінка результатів навчання:** чи вдалось навчити модель, чи немає перенавчання, чи не потрібно щось змінити в параметрах навчання та інші думки щодо результатів, чи добре підходить модель для задачі/датасету, як можна покращити результати у вашому випадку.

Ми вважаємо, що для отримання максимально достовірних результатів прогнозування нам не вдалось навчити модель належним чином: точність на рівні 85% ще залишає простір для її покращення на 10-13%. На графіку функції втрат першого рішення можна побачити, що після 150 епох прослідковується малопомітна тенденція збільшення значень loss, що може свідчити про перенавчання, але приблизно після 225-230 епох крива функції втрат виходить на попередні показники.

Першим кроком до покращення результатів навчання ми бачимо отримання достовірних даних про реальну точність роботи моделі. Для цього ми маємо порівняти результати прогнозування, отримані за допомогою розроблених рішень, та існуючі. Це легко зробити, доповнивши програмний код підрахунком кількості вірних прогнозів та підсумувавши їх діаграмою. Таким чином ми зможемо отримати відповідь на запитання, як оцінювана точність співвідноситься з результативністю прогнозування. Реальні результати можуть виявитись достатньо якісними та співпадати в більшості випадків. У цьому випадку, не дивлячись на отримані не найкращі показники точності, ми зможемо стверджувати про коректність роботи моделі, та рекомендувати її для вирішення подібних задач. У гіршому випадку, якщо прогнози часто не співпадають з реальними результатами, ми зрозуміємо, що нам слід експериментувати з параметрами моделі або розглянути інші методи аналізу даних.

Багатошарові персептрони довели можливість знаходити приблизні рішення для дуже складних задач. Вони є універсальними апроксиматорами функцій, тому з успіхом використовуються при побудові регресійних моделей. Оскільки класифікація може вважатись частковим випадком регресії, на основі MLP можна будувати класифікатори. З цього виходить, що цей інструмент підходить під рішення нашої задачі, в основі якої лежить категоризація. Але, інші методи, які в першу чергу призначені для вирішення задач класифікації, могли б виявитись більш ефективними. До них належать логістична регресія, алгоритм k-найближчих сусідів, випадковий ліс, метод опорних векторів, метод головних компонент тощо. Можливо, деякі з цих методів могли б більше підійти для вирішення нашої задачі.