به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق

تمرین چهارم

پرسش ۱	نام و نام خانوادگی	حمیدرضا نادی مقدم
	شماره دانشجویی	۸۱۰۱۰۳۲۶۴
پرسش ۲	نام و نام خانوادگی	علی صفری
	شماره دانشجویی	۸۱۰۲۰۲۱۵۳
	مهلت ارسال پاسخ	**۱۴۰۳.۰۹.۲۹**

فهرست

شکل‌ها

جدول‌ها

مقدمه

برای پیاده‌سازی پروژه از بستر Google Colab به منظور کد نویسی و اجرا استفاده شده است. تمامی مراحل کد و اجرای آن در این گزارش به تفصیل شرح داده شده است.

کد های نوشته شده همگی در پوشه‌ی Code و با پسوند ipynd. ذخیره شده است.

پرسش ۱. تشخیص هرزنامه

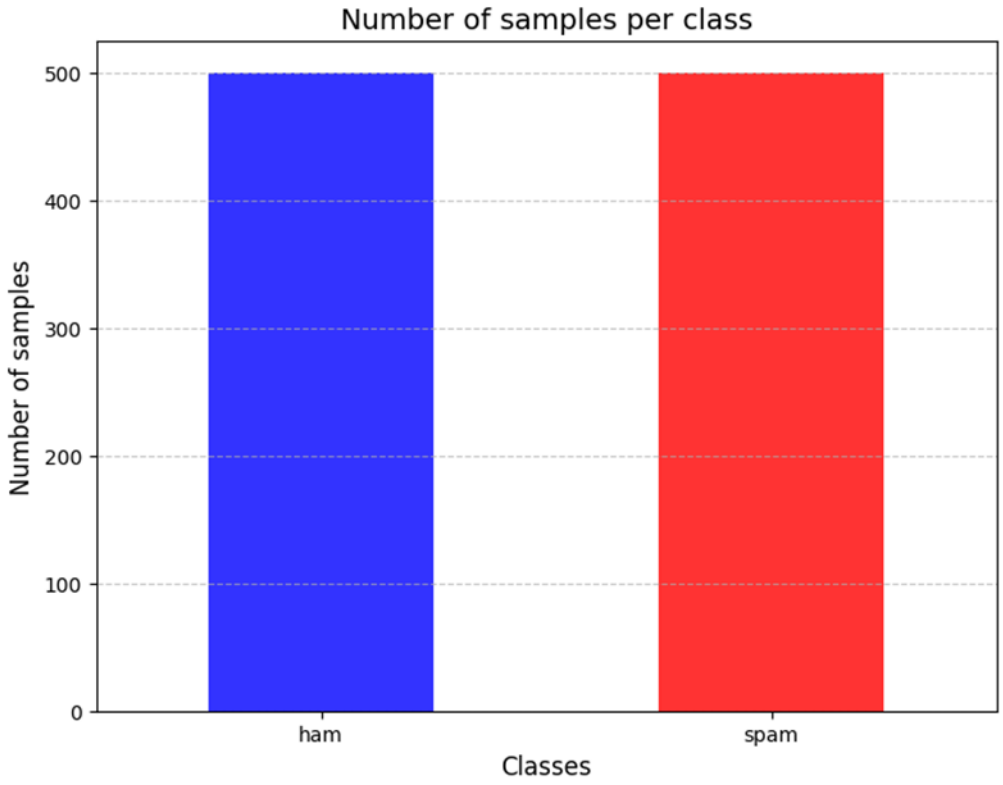
۱-۱. مجموعه داده

در ابتدا مجموعه داده را بارگذاری و فایل emails.csv را که شامل ست داده‌ها است را باز می‌کنیم و سپس تعداد نمونه‌های هر کلاس را در نمودار میله‌ای رسم کرده‌ایم.

```
data = pd.read_csv(file_path)

plt.figure(figsize=(8, 6))
class_counts.plot(kind='bar', color=['blue', 'red'], alpha=0.8)
plt.title('Number of samples per class', fontsize=14)
plt.xlabel('Classes', fontsize=12)
plt.ylabel('Number of samples', fontsize=12)
plt.xticks(rotation=0, fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

شکل ۱- کد بارگذاری و نمایش نمودار میله‌ای کلاس‌ها



شکل ۲ - نمودار میله‌ای تعداد هر کلاس

۱-۲. پیش‌پردازش داده‌ها

پیش‌پردازش متن در پردازش زبان طبیعی (NLP) یک مرحله‌ی رایج برای آماده‌سازی داده‌ها است که معمولاً برای هماهنگ سازی فرمت داده‌ها و حذف نویزها انجام می‌شود.

برای حذف URL ها از یک عبارت منظم (Regex) برای حذف آدرس‌های وب استفاده شده است. و از همین عبارات منظم برای شناسایی و حذف تمامی آدرس ایمیل‌ها، شماره تلفن یا اعداد حدقل ۱۰ رقمی، و حروفی که بیش از سه بار تکرار شده‌اند و جایگزینی آن با یک بار آن استفاده شده است.

برای حذف کلمات توقف که معمولاً اطلاعات معنایی خاصی ندارند، از یک فایل که شامل تمامی این کلمات است استفاده شده است. در این فایل تمامی کلمات توقف و اضافه در یک سطر قرار داشته که همه آن به یک مجموعه (set) متغییر اضافه شده است و در صورتی که در متن چنین کلمه‌ای باشد آن را حذف می‌کند.

```python
with open('farsi.txt', 'r', encoding='utf-8') as file:
    persian_stopwords = set(line.strip() for line in file)

def preprocess_text(text):
    text = text.replace("\n", " ")
    text = re.sub(r"http\S+|www\S+|https\S+", "", text, flags=re.MULTILINE)
    text = re.sub(r'\S+@\S+', '', text)
    text = re.sub(r'\b\d{10,}\b', '', text)
    text = re.sub(r'(.)\1{2,}', r'\1', text)
    tokens = text.split()
    filtered_tokens = [word for word in tokens if word not in persian_stopwords]
    text = ' '.join(filtered_tokens)

    return text

data['processed_text'] = data['text'].apply(preprocess_text)
print("Some Processed Text")
print(data[['text', 'processed_text']].head(3))
```

```
Some Processed Text
                                                text  \
0  من پارسال اصلا آزاد شرکت نک.\nممنون آقا سامان...
1  بالاخره آزمونارشد تموم شد من\nسلام آقای کریمی...
2  درود بر حاج وحیدی بنده بعنوان یک دکتری تاریخ ...

                                      processed_text
0  ممنون آقا سامان. آزاد شرکت سراسری قبول نشدم. ...
1  سلام آقای کریمی آزمونارشد تموم شدم یکم راهنما...
2  درود حاج وحیدی بنده بعنوان دکتری تاریخ دستی ت...
```

شکل ۳- کد پیش پردازش متن

۱-۳. نمایش ویژگی‌ها

ابعاد پیش‌فرض بردار تعبیه در ParsBERT

بردارهای تعبیه در مدل ParsBERT دارای ابعاد ۷۶۸ هستند. این مقدار برای نسخه‌های مبتنی بر BERT پایه (base) است. اگر از نسخه‌های بزرگتر یا اصلاح شده استفاده شود (مانند BERT-large)، این عدد می‌تواند بیشتر باشد (معمولاً ۱۰۲۴).

تعداد ابعاد بردار بیانگر چیست؟

تعداد ابعاد بردار تعبیه نشان‌دهنده تعداد ویژگی‌های انتزاعی است که مدل برای توصیف معنایی و نحوی کلمات از آنها استفاده می‌کند. هر بعد به یکی از ویژگی‌های پنهانی زبانی اختصاص دارد، اما این ویژگی‌ها معمولا توسط مدل به طور خودکار یاد گرفته می‌شوند و معنای مشخصی به صورت مستقل ندارند.

به طور کلی این ابعاد شامل اطلاعات نحوی، معنایی، گرامری، و حتی موقعیتی کلمات در جمله هستند. و بردار تعبیه به عنوان یک نمایش عددی فشرده عمل می‌کند که می‌تواند معانی مشابه را برای کلمات مرتبط در فضای برداری نشان دهد.

به طور خلاصه هر بعد می‌تواند ویژگی خاصی را نشان دهد، مانند معنای کلمه، نقش نحوی، یا روابط معنایی آن با کلمات دیگر. ابعاد بیشتر به معنای افزایش قدرت مدل در نمایش جزئیات دقیق‌تر است، اما همچنین پیچیدگی محاسباتی و نیاز به حافظه را افزایش می‌دهد.

مفهوم بردار تعبیه

بردار تعبیه، یک نمایش عددی متراکم از کلمات است که به کمک مدل‌های یادگیری عمیق مانند ParsBERT تولید می‌شود. این بردارها به مدل کمک می‌کنند تا اطلاعات زبانی را به شکلی قابل استفاده برای الگوریتم‌های محاسباتی بیان کند.

یا بردار تعبیه (Embedding Vector)، نمایش عددی یک کلمه یا جمله در یک فضای چند بعدی است. این نمایش به گونه‌ای طراحی شده است که کلمات یا جملاتی که معنا، زمینه یا نقش مشابه‌ی دارند، در این فضا به یکدیگر نزدیک باشند. و از ویژگی‌های کلیدی بردار تعبیه می توان به فشرده‌سازی معنای کلمات در یک فضای عددی، امکان مقایسه معنایی بین کلمات بر اساس فاصله یا زاویه بین بردارها و کاربرد در پردازش زبان طبیعی (NLP) برای درک روابط معنایی اشاره کرد.

کلمات مشابه از نظر معنایی (مثلاً "خانه" و "مسکن") یا نحوی معمولاً بردارهایی نزدیک به هم در فضای برداری دارند. مدل‌های مانند ParsBERT از بافت (Context) کلمه در جمله استفاده می‌کنند؛ بنابراین کلماتی که در جملات مشابه به کار می‌روند، تعبیه‌های مشابهی خواهند داشت.

- کلمات مترادف: کلماتی مانند "زیبا" و "خوشگل" یا "بزرگ" و "عظیم".
- کلمات هم‌گروه معنایی: کلماتی مانند "مدرسه" و "دانشگاه" یا "ماشین" و "اتومبیل".
- کلماتی که در یک زمینه خاص استفاده می‌شوند: مثلا "پزشک"، "پرستار" و "بیمارستان" که در حوزه پزشکی قرار دارند.
- کلمات هم‌ریشه: مانند "کتاب" و "کتابخانه".

بردار تعبیه در ParsBERT یک نمایش غنی از معانی کلمات است که بر پایه زمینه زبانی تولید می‌شود. این بردارها ارتباط معنایی و نحوی کلمات را مدل‌سازی می‌کنند و به همین دلیل، کلمات با مفاهیم نزدیک در فضای برداری به یکدیگر نزدیک خواهند بود.

```python
tokenizer = AutoTokenizer.from_pretrained("HooshvareLab/bert-fa-zwnj-base")
model = AutoModel.from_pretrained("HooshvareLab/bert-fa-zwnj-base")

def tokenize_and_pad(text, max_length=32):
    tokens = tokenizer(text, padding='max_length', truncation=True, max_length=max_length, return_tensors='pt')
    return tokens

train_data['tokens'] = train_data['processed_text'].apply(lambda x: tokenize_and_pad(x, max_length=32))
val_data['tokens'] = val_data['processed_text'].apply(lambda x: tokenize_and_pad(x, max_length=32))
test_data['tokens'] = test_data['processed_text'].apply(lambda x: tokenize_and_pad(x, max_length=32))


def get_embeddings(tokens):
    with torch.no_grad():
        outputs = model(**tokens)
    return outputs.last_hidden_state.mean(dim=1).squeeze().numpy()

train_data['embeddings'] = train_data['tokens'].apply(get_embeddings)
val_data['embeddings'] = val_data['tokens'].apply(get_embeddings)
test_data['embeddings'] = test_data['tokens'].apply(get_embeddings)

pca = PCA(n_components=120)
embeddings = train_data['embeddings'].tolist() + val_data['embeddings'].tolist() + test_data['embeddings'].tolist()
reduced_embeddings = pca.fit_transform(embeddings)
train_data['reduced_embeddings'] = list(reduced_embeddings[:len(train_data)])
val_data['reduced_embeddings'] = list(reduced_embeddings[len(train_data):len(train_data)+len(val_data)])
test_data['reduced_embeddings'] = list(reduced_embeddings[len(train_data)+len(val_data):])
```

شکل ۴- کد برای توکن سازی و بردار تعبیه و کاهش آن

۱-۴. ساخت مدل

در زیر نقاط قوت و ضعف هر یک از مدل‌ها CNN و LSTM را به صورت مختصر توضیح داده‌ایم.

مدل شبکه‌های عصبی کانولوشنی (CNN)

نقاط قوت

- توانایی شناسایی الگوهای فضایی: CNN با استفاده از فیلترهای کانولوشنی، اطلاعات محلی مانند لبه‌ها، گوشه‌ها، و بافت‌ها را به‌خوبی استخراج می‌کند. این ویژگی در پردازش تصاویر و داده‌های مکانی بسیار عالی است.
- کاهش پیچیدگی مدل: لایه‌های Pooling (مانند MaxPooling) باعث کاهش ابعاد داده و تعداد پارامترها می‌شوند. این امر علاوه بر جلوگیری از بیش‌برازش (Overfitting)، محاسبات را سریعتر می‌کند.
- مقیاس‌پذیری بالا: معماری CNN به گونه‌ای طراحی شده که بخوبی در داده‌های دو یا سه بعدی (مانند تصاویر و ویدئوها) عمل می‌کند و می‌تواند الگوهای تکرارشونده را شناسایی کند.

نقاط ضعف:

- عدم درک روابط زمانی: CNN فقط روابط مکانی را مدل‌سازی می‌کند و قادر به درک ترتیب یا وابستگی بین داده‌ها در طول زمان نیست. به‌عنوان مثال، نمی‌تواند توالی حرکت در یک ویدئو یا روند تغییر در سری زمانی را درک کند.
- نیاز به داده‌های ساختاریافته: CNN برای داده‌هایی مانند تصاویر که ساختار منظم دارند، مناسب است و در داده‌های غیرساختاریافته مانند متن خام یا توالی‌های عددی ضعیف عمل می‌کند.

مدل LSTM:

نقاط قوت:

- حافظه بلندمدت و کوتاه‌مدت: LSTM از معماری دروازه‌ها (Gates) استفاده می‌کند که به آن اجازه می‌دهد اطلاعات مهم را ذخیره کرده و اطلاعات غیرضروری را حذف کند. این ویژگی باعث می‌شود برای داده‌های دنباله‌ای (مانند متن یا صوت) که اطلاعات گذشته در تحلیل آینده مهم هستند، مناسب باشد.
- مدیریت داده‌های متغیر طول: برخلاف CNN، LSTM می‌تواند با داده‌هایی که طول آنها متغیر است (مانند جملات کوتاه و بلند) کار کند.
- تحلیل وابستگی‌های زمانی: LSTM در مدل‌سازی الگوهایی که در طول زمان تغییر می‌کنند (مانند پیش‌بینی قیمت سهام) بسیار موثر است.

نقاط ضعف:

- محاسبات پیچیده‌تر: فرآیندهای دروازه‌ای LSTM (ورودی، خروجی و فراموشی) نیاز به محاسبات بیشتری دارند که باعث کاهش سرعت و افزایش مصرف منابع می‌شود.
- عدم استخراج ویژگی‌های فضایی: LSTM نمی‌تواند اطلاعات مکانی یا ساختاریافته را بدون پیش‌پردازش استخراج کند. برای مثال، نمی‌تواند الگوهای موجود در تصاویر را شناسایی کند.

چرا CNN و LSTM را با هم ادغام می‌کنند.

هدف از ترکیب این دو مدل یعنی ادغام CNN و LSTM امکان بهره‌برداری از قدرت پردازش فضایی CNN و توانایی تحلیل روابط زمانی LSTM را فراهم می‌کند. این ترکیب می‌تواند مشکلاتی را که هر یک

از این مدل‌ها به‌تنهایی در حل آنها ناتوان هستند، برطرف کند. CNN به‌عنوان یک استخراج‌کننده ویژگی‌ها عمل می‌کند. و LSTM به‌عنوان مدل‌کننده روابط زمانی یا ترتیبی استفاده می‌شود.

یا به عبارتی دیگر ادغام این دو مدل با هدف بهره‌گیری از نقاط قوت هر دو و جبران ضعف‌های آنها انجام می‌شود. اهداف مثل پردازش داده‌های ترکیبی (فضایی-زمانی) که برای داده‌هایی که هم جنبه‌های فضایی (مانند ویژگی‌های محلی در تصاویر) و هم جنبه‌های زمانی یا ترتیبی (مانند تغییرات در طول زمان) دارند، ترکیب CNN و LSTM ایده‌آل است. مثل تحلیل ویدئو (شناسایی اشیاء در فریم‌ها با CNN و تحلیل ترتیب فریم‌ها با LSTM). هدف دیگر استخراج ویژگی‌های فضایی توسط CNN است. CNN ویژگی‌های مهم و فشرده را از داده‌های فضایی استخراج کرده و به LSTM ارائه می‌دهد. دیگری هدف آن مدیریت وابستگی‌های زمانی توسط LSTM است. LSTM دنباله زمانی ویژگی‌های استخراج‌شده توسط CNN را تحلیل می‌کند. و دیگری هدف آن افزایش دقت در داده‌های چند بعدی است. این ترکیب در کاربردهایی مانند پردازش ویدئو، تحلیل سری‌های زمانی تصویری (مانند MRI)، و ترجمه ماشینی تصویری بسیار مؤثر است.

چند کاربردهای عملی ترکیب CNN-LSTM:

پردازش ویدئو: که هدف شناسایی حرکات یا رویدادها در یک ویدئو. و عملکرد آن به این صورت است که CNN اطلاعات هر فریم را استخراج می‌کند و LSTM ترتیب زمانی این فریم‌ها را تحلیل می‌کند.

تحلیل سری‌های زمانی چندبعدی: هدف در این کار پیش‌بینی روندها یا تشخیص الگوها. و عملکرد آن این است که CNN ویژگی‌های مکانی (مثلاً داده‌های حسگرهای مختلف) را استخراج کرده و LSTM تغییرات زمانی این ویژگی‌ها را مدل‌سازی می‌کند.

و در از چند کاربرد دیگر آن می توان به ترجمه ماشینی تصویری، تشخیص صدا و گفتار اشاره کرد.

```python
def build_model(learning_rate, optimizer):
    model = Sequential([
        Embedding(input_dim=10000, output_dim=120, input_length=32),
        Conv1D(64, 3, activation='relu'),
        MaxPooling1D(2),
        LSTM(64, return_sequences=False),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    if optimizer == 'Adam':
        opt = Adam(learning_rate=learning_rate)
    elif optimizer == 'SGD':
        opt = SGD(learning_rate=learning_rate)
    model.compile(optimizer=opt, loss='binary_crossentropy', metrics=['accuracy'])
    return model


batch_sizes = [8, 64]
learning_rates = [0.001, 0.0001]
optimizers = ['Adam', 'SGD']

best_model = None
best_val_accuracy = 0
best_learning_rate = 0
best_optimizer = ''
best_batch_size = 0

for batch_size in batch_sizes:
    for learning_rate in learning_rates:
        for optimizer in optimizers:
            print(f"Training with batch_size={batch_size}, learning_rate={learning_rate}, optimizer={optimizer}")
            model = build_model(learning_rate, optimizer)
            early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)
            model.fit(
                x=torch.tensor(train_data['reduced_embeddings'].tolist()),
                y=train_data['label'].values,
                validation_data=(torch.tensor(val_data['reduced_embeddings'].tolist()), val_data['label'].values),
                batch_size=batch_size,
                epochs=10,
                callbacks=[early_stopping],
                verbose=0
            )
            val_acc = model.evaluate(torch.tensor(val_data['reduced_embeddings'].tolist()), val_data['label'].values, verbose=0)[1]
            if val_acc > best_val_accuracy:
                best_val_accuracy = val_acc
                best_model = model
                best_learning_rate = learning_rate
                best_optimizer = optimizer
                best_batch_size = batch_size

print(f"Best hyperparameters: batch_size={best_batch_size}, learning_rate={best_learning_rate}, optimizer={best_optimizer}")
```

شکل ۵- کد پیاده سازی الگوریتم حریصانه برای یافتن بهترین هایپر پارامترها

با استفاده از الگوریتم حریصانه پارامترهای بهینه ساز Adam، نرخ یادگیری ۰.۰۰۱ و اندازه دسته ۸ بدست آمد.

```python
def build_cnn_lstm_model(optimizer=Adam, learning_rate=0.001):
    model = Sequential([
        Embedding(input_dim=10000, output_dim=120, input_length=32),
        Conv1D(64, 3, activation='relu'),
        MaxPooling1D(2),
        LSTM(64, return_sequences=False),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=optimizer(learning_rate=learning_rate), loss='binary_crossentropy', metrics=['accuracy'])
    return model

def build_cnn_model(optimizer=Adam, learning_rate=0.001):
    model = Sequential([
        Embedding(input_dim=10000, output_dim=120, input_length=32),
        Conv1D(64, 3, activation='relu'),
        MaxPooling1D(2),
        Flatten(),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=optimizer(learning_rate=learning_rate), loss='binary_crossentropy', metrics=['accuracy'])
    return model

def build_lstm_model(optimizer=Adam, learning_rate=0.001):
    model = Sequential([
        Embedding(input_dim=10000, output_dim=120, input_length=32),
        LSTM(64, return_sequences=False),
        Dense(64, activation='relu'),
        Dropout(0.5),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer=optimizer(learning_rate=learning_rate), loss='binary_crossentropy', metrics=['accuracy'])
    return model


cnn_model = build_cnn_model(optimizer=globals()[best_optimizer], learning_rate=best_learning_rate)
cnn_model.fit(
    x=np.array(train_data['reduced_embeddings'].tolist()),
    y=train_data['label'].values,
    validation_data=(np.array(val_data['reduced_embeddings'].tolist()), val_data['label'].values),
    batch_size=64,
    epochs=10,
    verbose=1
)

lstm_model = build_lstm_model(optimizer=globals()[best_optimizer], learning_rate=best_learning_rate)
lstm_model.fit(
    x=np.array(train_data['reduced_embeddings'].tolist()),
    y=train_data['label'].values,
    validation_data=(np.array(val_data['reduced_embeddings'].tolist()), val_data['label'].values),
    batch_size=64,
    epochs=10,
    verbose=1
)

cnn_lstm_model = build_cnn_lstm_model(optimizer=globals()[best_optimizer], learning_rate=best_learning_rate)
cnn_lstm_model.fit(
    x=np.array(train_data['reduced_embeddings'].tolist()),
    y=train_data['label'].values,
    validation_data=(np.array(val_data['reduced_embeddings'].tolist()), val_data['label'].values),
    batch_size=64,
    epochs=10,
    verbose=1
)


print("CNN Accuracy: ", cnn_model.evaluate(np.array(test_data['reduced_embeddings'].tolist()), test_data['label'].values)[1])
print("LSTM Accuracy:", lstm_model.evaluate(np.array(test_data['reduced_embeddings'].tolist()), test_data['label'].values)[1])
print("CNN-LSTM Accuracy:", cnn_lstm_model.evaluate(np.array(test_data['reduced_embeddings'].tolist()), test_data['label'].values)[1])
```

شکل ۶- پیاده سازی و آموزش مدل **LSTM ،CNN** و ادغامی

۱-۵. ارزیابی

با استفاده از داده‌های تست هر سه مدل را ارزیابی کرده و معیارهای آن را بدست آوردیم. نتایج معیارهای مختلف برای هر مدل را در جدول ۱ لیست کرده‌ایم. چون تعداد داده‌ها برای این نوع شبکه‌ها بسیار کم است لذا دقت بدست آمده نیز به نسبت پایین است.

```
def evaluate_model(model, x_test, y_test):
    predictions = (model.predict(x_test) > 0.5).astype(int)
    acc = accuracy_score(y_test, predictions)
    prec = precision_score(y_test, predictions)
    rec = recall_score(y_test, predictions)
    f1 = f1_score(y_test, predictions)
    auc = roc_auc_score(y_test, model.predict(x_test))
    return acc, prec, rec, f1, auc

x_test = np.array(test_data['reduced_embeddings'].tolist())
y_test = test_data['label'].values

cnn_metrics = evaluate_model(cnn_model, x_test, y_test)
lstm_metrics = evaluate_model(lstm_model, x_test, y_test)
cnn_lstm_metrics = evaluate_model(cnn_lstm_model, x_test, y_test)

results = pd.DataFrame({
    'Model': ['CNN', 'LSTM', 'CNN-LSTM'],
    'Accuracy': [cnn_metrics[0], lstm_metrics[0], cnn_lstm_metrics[0]],
    'Precision': [cnn_metrics[1], lstm_metrics[1], cnn_lstm_metrics[1]],
    'Recall': [cnn_metrics[2], lstm_metrics[2], cnn_lstm_metrics[2]],
    'F1-Score': [cnn_metrics[3], lstm_metrics[3], cnn_lstm_metrics[3]],
    'AUC': [cnn_metrics[4], lstm_metrics[4], cnn_lstm_metrics[4]]
})

print(results)
```

شکل ۷- کد ارزیابی هر سه مدل

جدول ۱- نتایج الگوریتم های طبقه بندی **LSTM ،CCN** و ادغامی

مدل	Accuracy	Precision	Recall	F1-Score	ROC AUC
CNN	0.853333	0.920635	0.773333	0.840580	0.923067
LSTM	0.500000	0.000000	0.000000	0.000000	0.600867
CNN-LSTM	0.670000	0.673469	0.660000	0.666667	0.708089

در این قسمت همه مدل‌های سنتی که در مقاله ذکر شده بود را با استفاده از کتابخانه پیاده‌سازی و سپس آموزش دادیم و با استفاده از داده‌های تست مورد ارزیابی قرار دادیم و سپس معیارهای Accuracy ، Precision، Recall، F1-Score و AUC را برای هر مدل محاسبه کردیم و در جدول ۲ گزارش داده‌ایم.

```
vectorizer = CountVectorizer(max_features=1000)
X_train = vectorizer.fit_transform(train_data['processed_text']).toarray()
X_val = vectorizer.transform(val_data['processed_text']).toarray()
X_test = vectorizer.transform(test_data['processed_text']).toarray()

y_train = train_data['label'].values
y_val = val_data['label'].values
y_test = test_data['label'].values

models = {
    "Logistic Regression": LogisticRegression(),
    "SVM": SVC(probability=True),
    "Random Forest": RandomForestClassifier(),
    "Naive Bayes": MultinomialNB(),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "AdaBoost": AdaBoostClassifier(),
    "Bagging Classifier": BaggingClassifier(),
    "Extra Trees": ExtraTreesClassifier()
}

results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    probabilities = model.predict_proba(X_test)[:, 1] if hasattr(model, 'predict_proba') else None

    acc = accuracy_score(y_test, predictions)
    prec = precision_score(y_test, predictions)
    rec = recall_score(y_test, predictions)
    f1 = f1_score(y_test, predictions)
    auc = roc_auc_score(y_test, probabilities) if probabilities is not None else None

    results.append({
        'Model': name,
        'Accuracy': acc,
        'Precision': prec,
        'Recall': rec,
        'F1-Score': f1,
        'AUC': auc
    })

results = pd.DataFrame(results)
print(results)
```

شکل ۸- پیاده‌سازی، آموزش و ارزیابی مدل‌های سنتی

جدول ۲ - نتایج الگوریتم‌های طبقه بندی

	Accuracy	Precision	Recall	F1-Score	ROC AUC
CNN	0.853333	0.920635	0.773333	0.840580	0.923067
LSTM	0.500000	0.000000	0.000000	0.000000	0.600867
CNN-LSTM	0.670000	0.673469	0.660000	0.666667	0.708089
Logistic Regression	0.950000	**1.000000**	0.900000	0.947368	**0.991022**
SVM	0.886667	0.975410	0.793333	0.875000	0.983867
Random Forest	0.943333	0.978417	0.906667	0.941176	0.986778
Naive Bayes	**0.953333**	0.972222	**0.933333**	0.952381	0.973467
K-Nearest Neighbors	0.800000	**1.000000**	0.600000	0.750000	0.901244
Decision Tree	0.920000	0.931507	0.906667	0.918919	0.920000
AdaBoost	0.933333	0.985075	0.880000	0.929577	0.975422
Bagging Classifier	0.923333	0.970370	0.873333	0.919298	0.978822
Extra Trees	0.950000	0.985612	0.913333	**0.948097**	0.990178

پرسش۲-پیشبینی ارزش نفت

۲-۱. مقدمه

برای حل این سوال از کتابخانه tensorflow برای مدل‌های deep learning و از pmdarima برای ARIMA استفاده شده است. کد مربوط به بخش پیاده‌سازی مدل‌های یادگیری عمیق در فایل HW4_2_Extra و کد مربوط به بخش ARIMA در فایل HW4_2_ARIMA ذخیره شده است.

۲-۲. مجموعه دادگان و آماده‌سازی

برای دانلود دیتای مورد نیاز از کتابخانه yfinance استفاده شده است. ابتدا دیتا در بازه‌ی زمانی مدنظر سوال استخراج شده و سپس ستون adj close به عنوان ویژگی اصلی مورد نظر قرار گرفته است.

```
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.arima.model import ARIMA
from pmdarima import auto_arima
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, GRU, Dense, Bidirectional
from tensorflow.keras.optimizers import Adam
from datetime import datetime
import tensorflow as tf

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)
```

شکل ۹- استخراج کتابخانه‌های مورد نیاز

```python
# Download the data for CL=F from Yahoo Finance starting from 2010
data = yf.download('CL=F', start='2010-01-01', end=datetime.now().strftime('%Y-%m-%d'))

# Focus on Adj Close column as per instructions
data = data[['Adj Close']]

# Check data structure and initial info
print("Initial Data Info:")
print(data.info())
print(data.head())
```

```
[*********************100%***********************]  1 of 1 completedInitial Data Info:
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3762 entries, 2010-01-04 to 2024-12-16
Data columns (total 1 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   (Adj Close, CL=F)  3762 non-null   float64
dtypes: float64(1)
memory usage: 58.8 KB
None
Price       Adj Close
Ticker           CL=F
Date
2010-01-04  81.510002
2010-01-05  81.769997
2010-01-06  83.180000
2010-01-07  82.660004
2010-01-08  82.750000
```

شکل ۱۰ - استخراج دیتا از سال ۲۰۱۰ و در نظر گرفتن ستون **Adj Close**

در گام بعد دیتای null و ۱۰ درصد داده‌ها به صورت رندوم حذف شده اند.

```python
# 2.1 Introduce an additional 10% random missing values into the existing data
non_null_indices = data.dropna().index
indices_to_nullify = np.random.choice(non_null_indices,
                                      size=int(len(non_null_indices) * 0.1),
                                      replace=False)
data.loc[indices_to_nullify] = np.nan
```

شکل ۱۱ - حذف رندوم ۱۰ درصد

در گام بعد نیاز به تکمیل داده‌ها داریم. با توجه به اینکه مقداری از داده ها null بوده و خود ما هم ۱۰ درصد حذف کردیم نیاز داریم کامل کنیم. در ادامه روش‌های رایج این کار را شرح داده و یکی از آنها را اجرا می‌کنیم.

۱. روش‌های حذف

- *حذف کامل سطر* :حذف سطرهایی که دارای مقادیر ناموجود هستند.
- *حذف ستون* :حذف ستون‌هایی که درصد زیادی از داده‌های آن گمشده است.

۲. **جایگزینی با میانگین، میانه یا مد**

- جایگزینی مقادیر گمشده با میانگین (عددی)، میانه (عددی) یا مد (دسته‌ای).

۳. **درون‌یابی خطی**

- استفاده از روش‌های خطی برای برآورد مقادیر گمشده بین نقاط داده موجود.

۴. **درون‌یابی همسایگان نزدیک(KNN)**

- استفاده از همسایگان نزدیک برای جایگزینی داده‌های گمشده.

۵. **پیش‌بینی مقادیر با رگرسیون**

- استفاده از مدل‌های رگرسیون برای پیش‌بینی مقادیر گمشده.

۶. **روش‌های پیشرفته**

- *درون‌یابی چندگانه* :تولید چند مقدار برای داده‌های گمشده و ترکیب نتایج.
- *شبکه‌های عصبی عمیق* :استفاده از یادگیری عمیق برای پیش‌بینی مقادیر گمشده.

برای پیاده‌سازی روش درونیابی خطی را استفاده کردیم.

```
# 2.2 Handle missing values
# The project requires proposing and implementing methods to fill missing data.
# We choose linear interpolation as a straightforward imputation method.
data_filled = data.interpolate(method='linear', limit_direction='both')
```

شکل ۱۲ - درونیابی خطی

در گام بعد به سراغ تقسیم داده‌ها به دو بخش آموزش و تست می‌رویم و آنها را نرمال می‌کنیم. طبق مقاله به نسبت ۷۰ درصد آموزش و ۳۰ درصد تست تقسیم کردیم. برای نرمال کردن هم مطابق مقاله از روش MinMaxScaler استفاده کرده‌ایم.

```python
# 2.4 Normalize the data
scaler = MinMaxScaler()
data_normalized = pd.DataFrame(
    scaler.fit_transform(data_filled),
    columns=data_filled.columns,
    index=data_filled.index
)

# 2.5 Split data into training (70%) and testing (30%)
train_size = int(len(data_normalized) * 0.7)
train_data = data_normalized.iloc[:train_size]
test_data = data_normalized.iloc[train_size:]

print("\nData split:")
print(f"Training set size: {len(train_data)}")
print(f"Testing set size: {len(test_data)}")
```

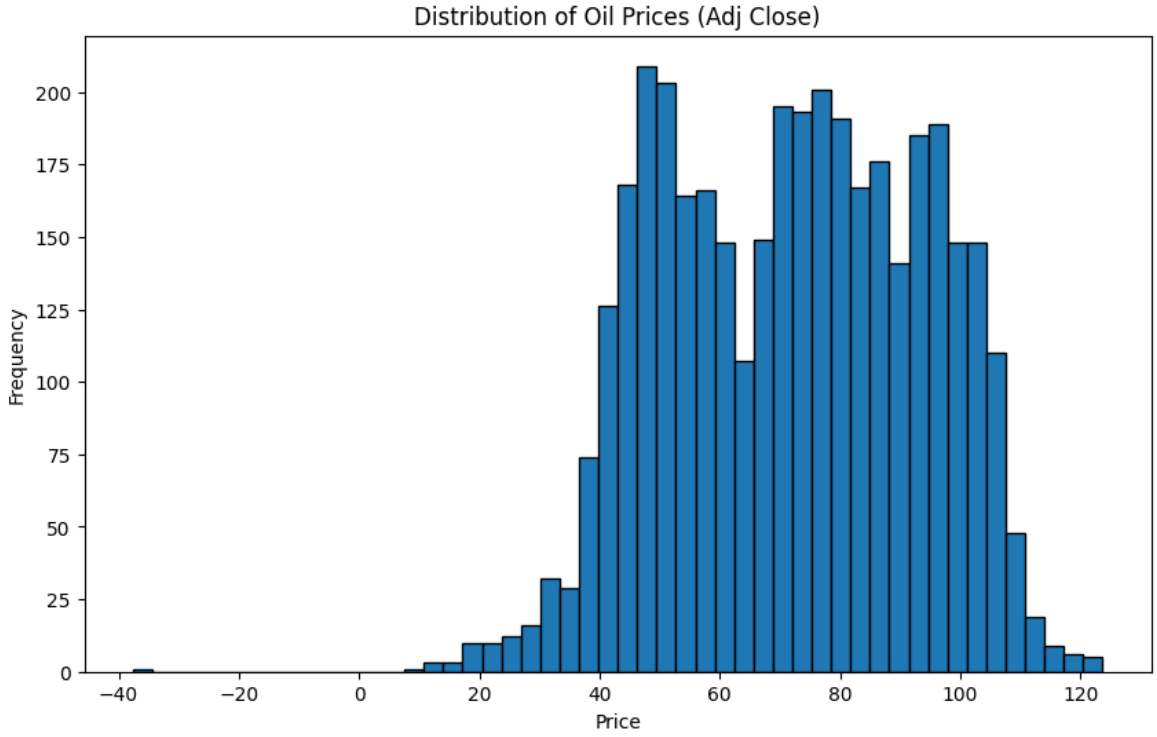
```
Data split:
Training set size: 2633
Testing set size: 1129
```

شکل ۱۳ - تقسیم دیتا و نرمال کردن

در گام بعد توزیع قیمت را به صورت هیستوگرام نمایش دادیم.

```python
# 2.3 Show histogram of price distribution (similar to the figure in the paper)
plt.figure(figsize=(10, 6))
plt.hist(data_filled['Adj Close'].dropna(), bins=50, edgecolor='black')
plt.title('Distribution of Oil Prices (Adj Close)')
plt.xlabel('Price')
plt.ylabel('Frequency')
plt.show()
```

شکل ۱۴ - کد نمایش هیستوگرام قیمت



شکل ۱۵ - هیستوگرام توزیع قیمت

۲-۳. پیاده‌سازی مدل‌ها

۲-۳-۱. آموزش مدل

در این بخش از پروژه، پیش‌بینی سری زمانی با استفاده از سه مدل یادگیری عمیق به شرح زیر انجام می‌شود:

۱. **LSTM** (Long Short-Term Memory)
۲. **GRU** (Gated Recurrent Unit)
۳. **Bi-LSTM** (Bidirectional Long Short-Term Memory)

هدف، پیش‌بینی مقادیر آینده سری زمانی و ارزیابی عملکرد مدل‌ها با استفاده از معیار **میانگین مربعات خطا (MSE)** به عنوان تابع خطا است. هایپرپارامترها مطابق با **جدول ۴** مقاله پیاده‌سازی شده‌اند.

۲ .هایپرپارامترهای استفاده شده

طبق جدول ۴ مقاله، مقادیر هایپرپارامترها برای همه مدل‌ها به صورت زیر در نظر گرفته شده است:

- **Learning Rate**مقدار ۰.۰۰۱
- **Batch Size**برابر با ۱۰۰
- **Epochs**تعداد ۵۰ تکرار
- **Units**
  - برای مدل‌های **LSTM**و **GRU**برابر با ۵۱۲
  - برای مدل **Bi-LSTM**برابر با ۱۰۲۴ (۵۱۲ واحد برای هر جهت)
- **Optimizer**: الگوریتمAdam

البته در مقاله هایپرپارامترهای دیگری هم در ابتدا بکار گرفته شده است اما در نهایت این مقادیر به عنوان مقادیر بهینه و نهایی در جدول چهار گزارش شده است.

یکی دیگر از هایپرپارامتر های بسیار با اهمیت seq_length است که در مقاله و صورت سوال عدد مشخصی برای آن انتخاب نشده است. اما در بخش related work اشاره شده که مقدار time step برابر ۲ نتایج خوبی داشته است. در ادامه به توضیح این هایپرپارامتر و فرض خود در این سوال خواهیم پرداخت.

Sequence Length (seq_length) چیست؟

seq_lengthیا طول دنباله ورودی یک پارامتر در مدل‌های سری زمانی است که مشخص می‌کند چه تعداد نقاط داده گذشته برای پیش‌بینی نقطه بعدی در نظر گرفته شوند. در مدل‌های بازگشتی مانند **LSTM**و **GRU**، مدل نیاز دارد که مقادیر ورودی به صورت توالی از داده‌ها باشند.

چرا باید انتخاب شود؟

انتخاب seq_lengthاهمیت زیادی دارد زیرا:

۱. اگر seq_lengthخیلی کوتاه باشد :
   - مدل نمی‌تواند الگوهای بلندمدت در داده‌ها را یاد بگیرد.
   - اطلاعات کافی برای پیش‌بینی وجود نخواهد داشت.
۲. اگر seq_lengthخیلی بلند باشد :
   - مدل پیچیدگی بالایی خواهد داشت.

- آموزش مدل زمان بیشتری می‌برد و ممکن است داده‌های غیرضروری در نظر گرفته شوند.

بنابراین، انتخاب طول مناسب دنباله باید براساس **ماهیت داده‌ها** و **رفتار زمانی** آن‌ها صورت گیرد.

مکانیزم استفاده از seq_length

در مدل‌های سری زمانی مانند LSTM وGRU ، ابتدا داده‌ها به دنباله‌هایی از طول seq_lengthتقسیم می‌شوند. هر دنباله برای پیش‌بینی مقدار بعدی استفاده می‌شود:

مثال :اگر seq_length = 10باشد و داده‌ها شامل مقادیر زیر باشند:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

دنباله‌ها به صورت زیر ایجاد می‌شوند:

- ورودی 1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] خروجی: 11
- ورودی 2: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11] خروجی: 12

انتخاب seq_length در پروژه

در این پروژه من مقدار seq_lengthرا ابتدا برابر ۱۰ انتخاب کردم. دلیل این انتخاب:

۱. با توجه به ماهیت داده‌ها، الگوهای کوتاه‌مدت کافی برای پیش‌بینی نقطه بعدی فراهم می‌شوند.

۲. طول ۱۰ باعث تعادل بین پیچیدگی مدل و کارایی آن می‌شود و آموزش مدل زمان مناسبی دارد.

اما بعد از مشورت با TA و با توجه به متن مقاله که در بخشی time step برابر ۲ در نظر گرفته شده بود، این مقدار را برابر ۲ در نظر گرفته و مجدد آموزش و ارزیابی را دنبال کردم.

لازم به ذکر است فایلی که با seq_length برابر ۱۰ اجرا شده است هم به عنوان تکلیف اضافه در پیوست قرار خواهد گرفت.

```python
# Create sequences of length seq_length
seq_length = 2
def create_sequences(data, seq_length=2):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i+seq_length])
        y.append(data[i+seq_length])
    return np.array(X), np.array(y)

X_train, y_train = create_sequences(train_data.values, seq_length)
X_test, y_test = create_sequences(test_data.values, seq_length)

# Reshape input for neural networks (LSTM/GRU/Bi-LSTM require 3D input: [samples, timesteps, features])
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

شکل ۱۶- تعریف تابع برای ساخت **sequence**

مکانیزم استفاده از seq_length

۱. در ابتدا، داده‌ها به دنباله‌هایی از طول seq_lengthتقسیم می‌شوند.
   - در کد بالا، برای هر نقطه داده در موقعیت ، seq_lengthمقدار قبل از آن انتخاب می‌شود.
   - مقدار به عنوان مقدار خروجی (هدف) استفاده می‌شود.

۲. این فرآیند با استفاده از یک حلقه **for**اجرا می‌شود و به ازای هر گام زمانی، دنباله‌های جدید برای آموزش مدل ساخته می‌شوند.

۳. در انتها، داده‌های ورودی باید به شکل سه‌بعدی برای مدل‌های **LSTM**، **GRU**و **Bi-LSTM** تبدیل شوند، زیرا این مدل‌ها نیاز به ورودی‌هایی با شکل [samples, timesteps, features]دارند.

۳ .معماری مدل‌ها

۱. مدل LSTM

مدل LSTM شامل یک لایه LSTM با **۵۱۲ واحد** و یک لایه Dense برای خروجی نهایی است:

```python
def create_lstm_model():
    model = Sequential([
        LSTM(512, input_shape=(seq_length, 1)),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

شکل ۱۷ - ساخت مدل **LSTM**

۲. مدل GRU

مدل GRU مشابه مدل LSTM است و تنها تفاوت در استفاده از لایه GRU به جای LSTM است:

```python
def create_gru_model():
    model = Sequential([
        GRU(512, input_shape=(seq_length, 1)),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

شکل ۱۸ - ساخت مدل **GRU**

۳. مدل Bi-LSTM

در مدل Bi-LSTM، از لایه Bidirectional LSTM استفاده می‌شود که به صورت **دو جهته** عمل می‌کند. مجموع واحدها برابر با ۱۰۲۴ (۵۱۲ واحد در هر جهت) است:

```python
def create_bilstm_model():
    # Bi-LSTM: According to the paper, units=1024 means total.
    model = Sequential([
        Bidirectional(LSTM(512, input_shape=(seq_length, 1))),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')
    return model
```

شکل ۱۹ - ساخت مدل **Bi LSTM**

۴. تابع آموزش مدل‌ها

برای آموزش مدل‌ها از رویه زیر استفاده شد:

- داده‌ها به دو بخش **آموزشی** و **اعتبارسنجی (Validation)** تقسیم شدند. **(البته لازم به ذکر است که این بخش در صورت سوال و مقاله مستقیما اشاره نشده و الزامی به انجام این کار نبود اما برای آموزش بهتر و بنا به اصول اولیه و معمول آموزش، این تقسیم بندی انجام گرفته است.)**
- **۲۰ درصد از داده‌ها** برای اعتبارسنجی در طول آموزش استفاده شد.
- از **میانگین مربعات خطا (MSE)** به عنوان تابع خطا برای ارزیابی استفاده شد.
- می‌توانیم از **early stop** برای جلوگیری از اجرای طولانی و بیهوده استفاده کنیم اما به دلیل اینکه اجرای مدل زمان زیادی نمی‌برد و در مقاله و صورت سوال اشاره ای نشده بود از اضافه کردن این بخش خودداری شد.

```
def train_model(model_fn, X_train, y_train):
    model = model_fn()
    history = model.fit(
        X_train, y_train,
        epochs=50,
        batch_size=100,
        validation_split=0.2,  # 20% for validation
        verbose=1
    )
    return model, history
```

شکل ۲۰- تعریف تابع آموزش

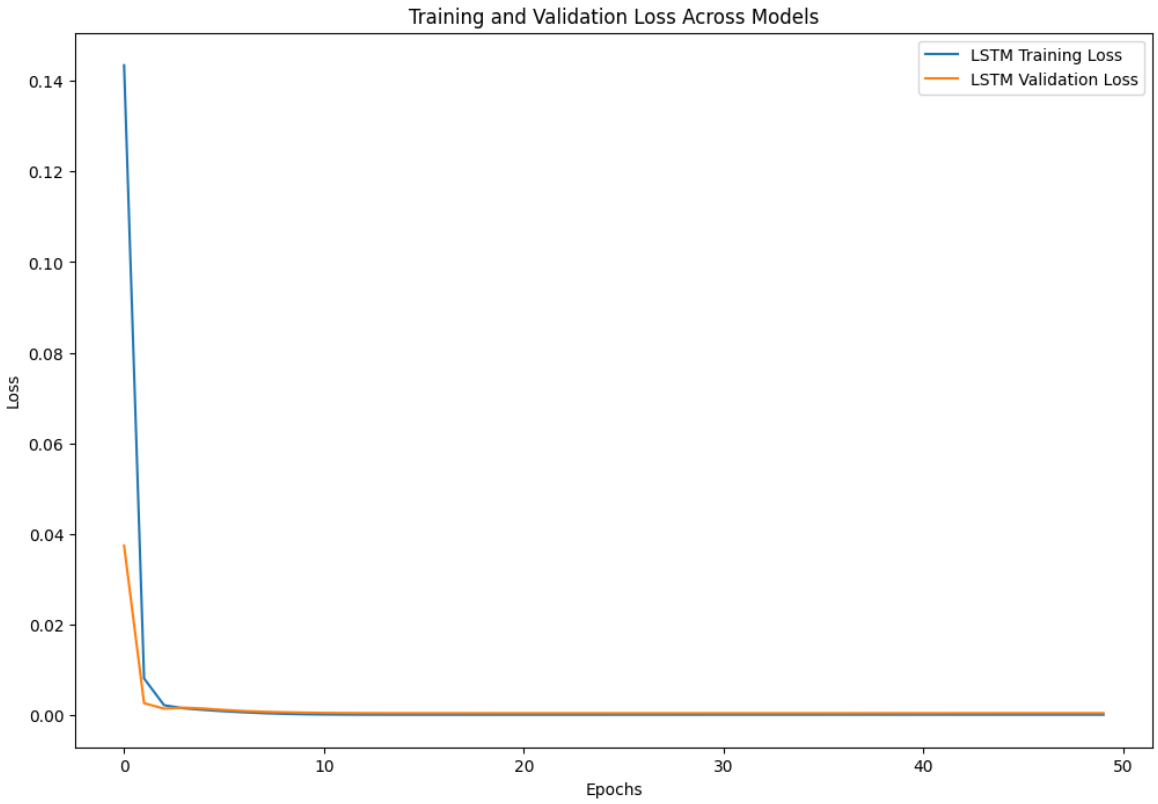
۵. فرایند آموزش مدل‌ها

در ادامه برای آموزش مدل هر مدل به صورت جداگانه با این تابع آموزش داده شد.

```
trained_models["LSTM"], history = train_model(create_lstm_model, X_train, y_train)
histories["LSTM"] = history
```

```
Epoch 22/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 227ms/step - loss: 2.0110e-04 - val_loss: 5.2334e-04
Epoch 23/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 309ms/step - loss: 1.9993e-04 - val_loss: 5.2063e-04
Epoch 24/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 253ms/step - loss: 1.9909e-04 - val_loss: 5.1824e-04
Epoch 25/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 7s 316ms/step - loss: 1.9836e-04 - val_loss: 5.1616e-04
Epoch 26/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 253ms/step - loss: 1.9769e-04 - val_loss: 5.1423e-04
Epoch 27/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 241ms/step - loss: 1.9707e-04 - val_loss: 5.1240e-04
Epoch 28/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 7s 314ms/step - loss: 1.9648e-04 - val_loss: 5.1064e-04
Epoch 29/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 5s 239ms/step - loss: 1.9592e-04 - val_loss: 5.0891e-04
Epoch 30/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 262ms/step - loss: 1.9540e-04 - val_loss: 5.0722e-04
Epoch 31/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 235ms/step - loss: 1.9490e-04 - val_loss: 5.0556e-04
Epoch 32/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 306ms/step - loss: 1.9444e-04 - val_loss: 5.0394e-04
Epoch 33/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 248ms/step - loss: 1.9402e-04 - val_loss: 5.0236e-04
```

شکل ۲۱ - بخشی از فرایند آموزش مدل **LSTM**

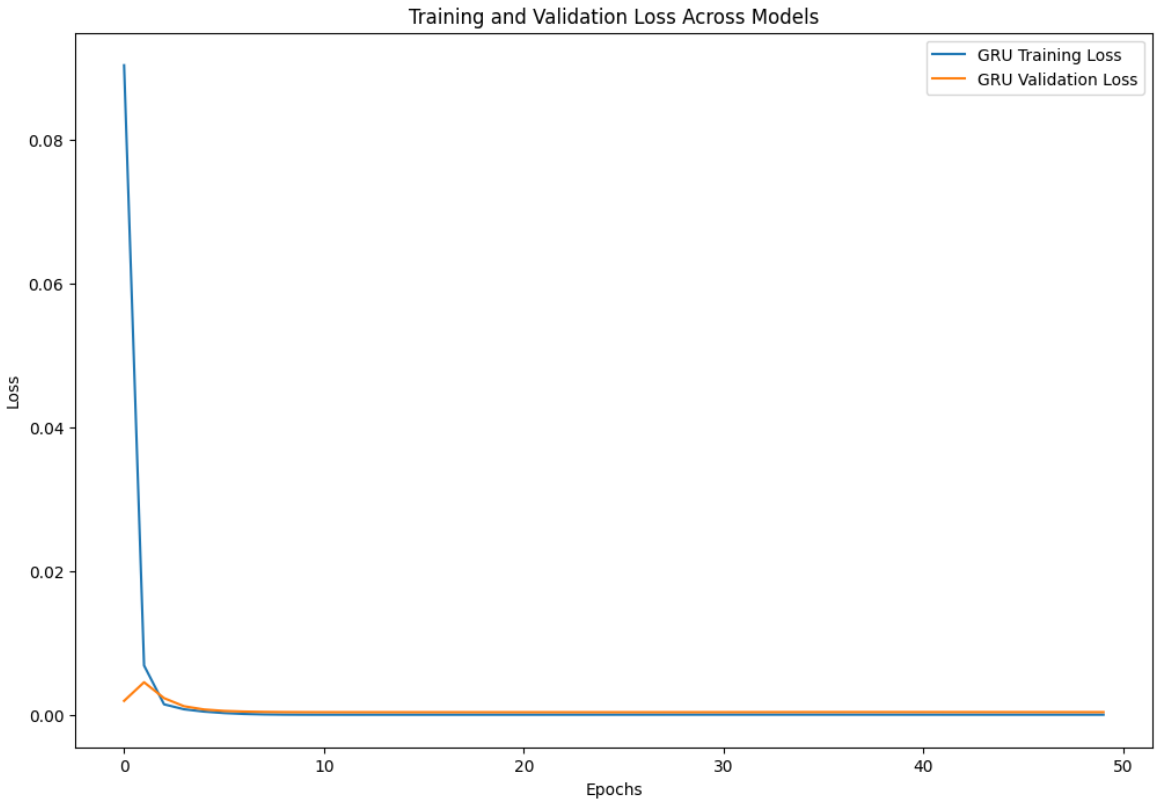


شکل ۲۲ - نمودار تغییرات **loss** مدل **LSTM**

```
print(f"\nTraining GRU...")
trained_models["GRU"], history = train_model(create_gru_model, X_train, y_train)
histories["GRU"] = history
```

```
Epoch 22/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 4s 171ms/step - loss: 1.2271e-04 - val_loss: 4.3223e-04
Epoch 23/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 7s 249ms/step - loss: 1.2234e-04 - val_loss: 4.3181e-04
Epoch 24/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 4s 174ms/step - loss: 1.2201e-04 - val_loss: 4.3140e-04
Epoch 25/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 5s 172ms/step - loss: 1.2170e-04 - val_loss: 4.3101e-04
Epoch 26/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 7s 248ms/step - loss: 1.2141e-04 - val_loss: 4.3063e-04
Epoch 27/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 176ms/step - loss: 1.2114e-04 - val_loss: 4.3026e-04
Epoch 28/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 211ms/step - loss: 1.2089e-04 - val_loss: 4.2991e-04
Epoch 29/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 4s 179ms/step - loss: 1.2066e-04 - val_loss: 4.2957e-04
Epoch 30/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 219ms/step - loss: 1.2045e-04 - val_loss: 4.2924e-04
Epoch 31/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 4s 170ms/step - loss: 1.2025e-04 - val_loss: 4.2893e-04
Epoch 32/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 5s 178ms/step - loss: 1.2008e-04 - val_loss: 4.2864e-04
Epoch 33/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 6s 238ms/step - loss: 1.1993e-04 - val_loss: 4.2836e-04
Epoch 34/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 4s 172ms/step - loss: 1.1981e-04 - val_loss: 4.2811e-04
Epoch 35/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 4s 173ms/step - loss: 1.1971e-04 - val_loss: 4.2787e-04
Epoch 36/50
```

شکل ۲۳ - بخشی از فرایند آموزش مدل **GRU**

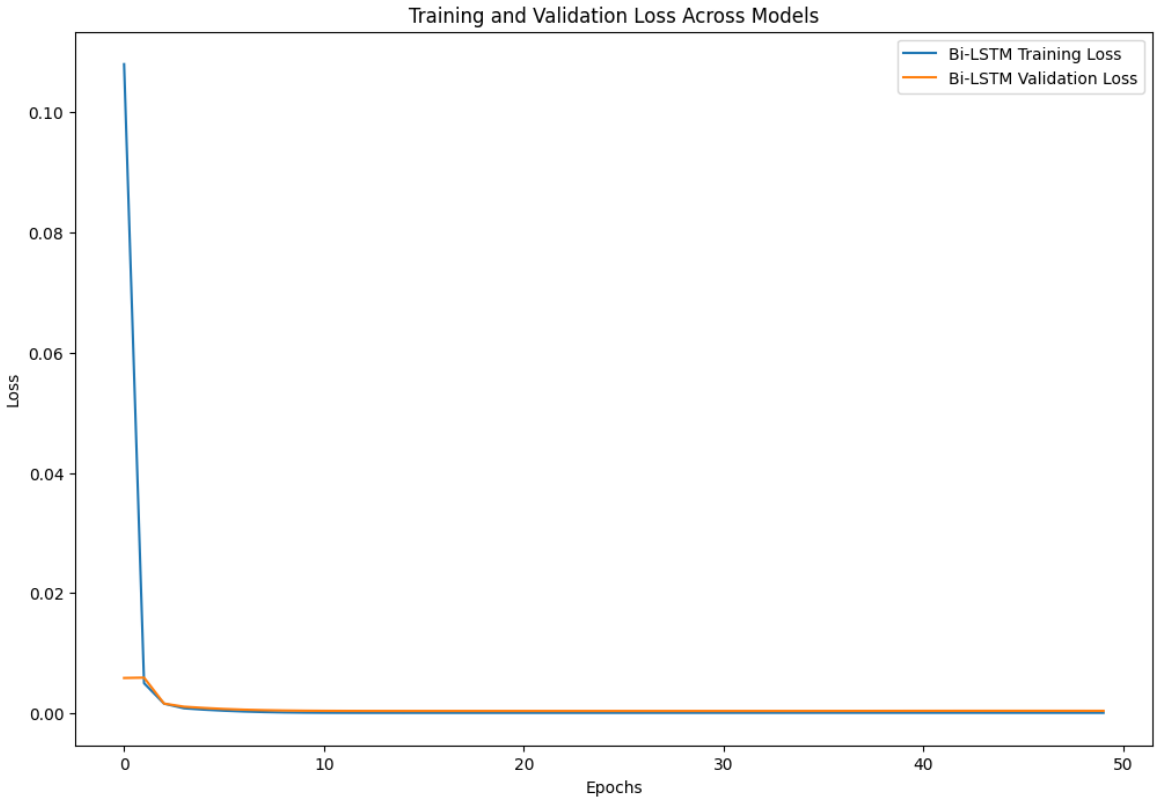


شکل ۲۴- نمودار تغییرات **loss** مدل **GRU**

```
print(f"\nTraining Bi-LSTM...")
trained_models["Bi-LSTM"], history = train_model(create_bilstm_model, X_train, y_train)
histories["Bi-LSTM"] = history
```

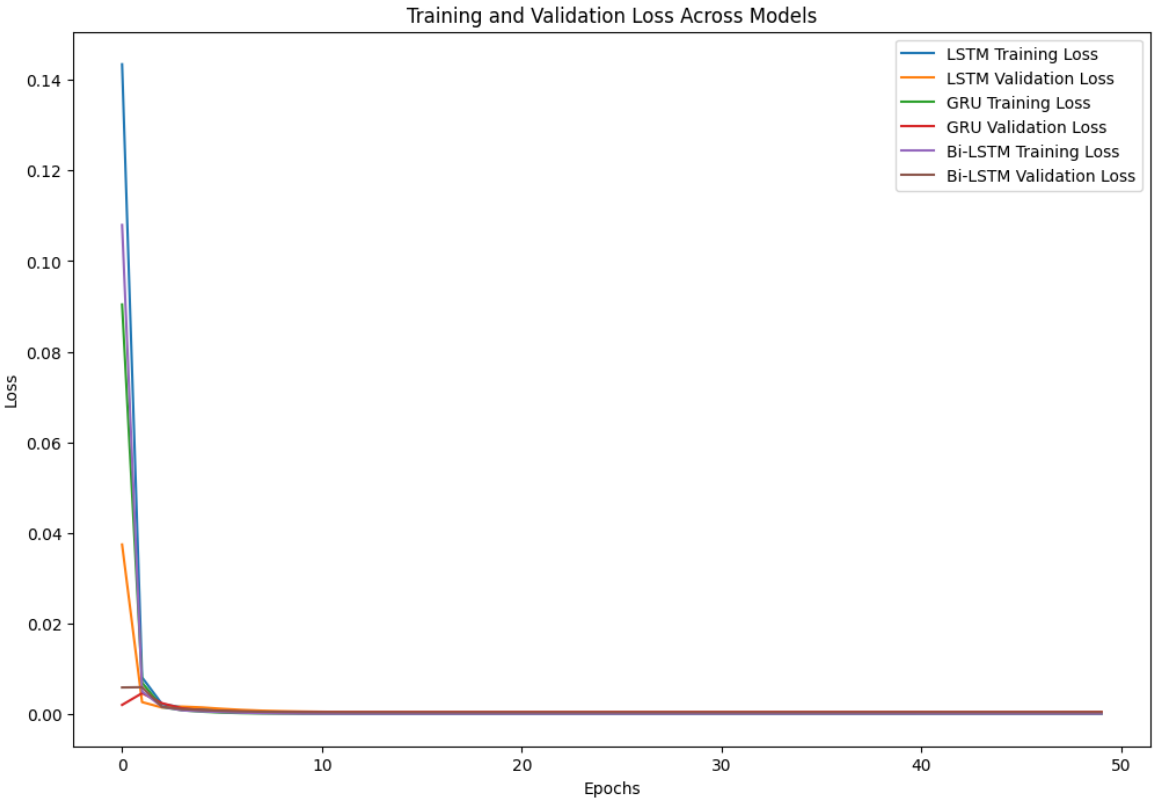
```
Epoch 22/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 495ms/step - loss: 2.5155e-04 - val_loss: 6.1856e-04
Epoch 23/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 523ms/step - loss: 2.5043e-04 - val_loss: 6.1939e-04
Epoch 24/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 20s 487ms/step - loss: 2.4940e-04 - val_loss: 6.2104e-04
Epoch 25/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 10s 474ms/step - loss: 2.4834e-04 - val_loss: 6.2243e-04
Epoch 26/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 533ms/step - loss: 2.4694e-04 - val_loss: 6.2130e-04
Epoch 27/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 20s 506ms/step - loss: 2.4486e-04 - val_loss: 6.1552e-04
Epoch 28/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 21s 530ms/step - loss: 2.4197e-04 - val_loss: 6.0574e-04
Epoch 29/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 20s 523ms/step - loss: 2.3869e-04 - val_loss: 5.9572e-04
Epoch 30/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 9s 449ms/step - loss: 2.3564e-04 - val_loss: 5.8830e-04
Epoch 31/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 515ms/step - loss: 2.3304e-04 - val_loss: 5.8297e-04
Epoch 32/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 534ms/step - loss: 2.3075e-04 - val_loss: 5.7817e-04
Epoch 33/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 19s 460ms/step - loss: 2.2859e-04 - val_loss: 5.7354e-04
Epoch 34/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 527ms/step - loss: 2.2647e-04 - val_loss: 5.6904e-04
Epoch 35/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 543ms/step - loss: 2.2438e-04 - val_loss: 5.6451e-04
Epoch 36/50
21/21 ━━━━━━━━━━━━━━━━━━━━ 11s 532ms/step - loss: 2.2234e-04 - val_loss: 5.5988e-04
```

شکل ۲۵- بخشی از فرایند آموزش مدل **Bi-LSTM**



شکل ۲۶- نمودار تغییرات **loss** مدل **Bi-LSTM**

همانطور که از نمودار loss مدل‌ها مشخص است همه آنها خیلی سریع به همگرایی رسیده اند اما با این حال ۵۰ ایپاک طی شد و از Validation هم برای بخش آموزش استفاده شد.



شکل ۲۷- نمودار تغییرات **loss** در تمام مدل‌ها در کنار هم[۱]

۲-۳-۲. نمایش نتایج

برای بررسی نتایج مدل و پیشبینی آن تابع evaluation نوشته شده است.

۱ .تعریف توابع ارزیابی

تابع evaluate_model

این تابع وظیفه دریافت داده‌های پیش‌بینی شده از مدل و محاسبه مقادیر معکوس نرمال‌سازی شده برای ارزیابی دارد:

۱ نمایش تغییرات loss به صورت اضافه و برای درک بهتر از فرایند آموزش قراره گرفته است.

```python
def evaluate_model(model, X, y, scaler):
    predictions = model.predict(X)
    predictions_inv = scaler.inverse_transform(predictions)
    actual_inv = scaler.inverse_transform(y.reshape(-1, 1))
    metrics = evaluate_predictions(actual_inv, predictions_inv)
    return metrics, predictions_inv, actual_inv

results = {}
predictions_dict = {}
if not trained_models:
    print("No models available to evaluate. Please train or save models first.")
else:
    results = {}
    predictions_dict = {}
    for model_name, model in trained_models.items():
        print(f"\nEvaluating {model_name}...")
        metrics, preds_inv, actual_inv = evaluate_model(model, X_test, y_test, scaler)
        results[model_name] = metrics
        predictions_dict[model_name] = preds_inv
        for k, v in metrics.items():
            print(f"{k}: {v:.4f}")
```

شکل ۲۸- تعریف تابع ارزیابی و بکارگیری آن

پیش‌بینی داده‌های تست :با استفاده از مدل آموزش دیده بر روی داده‌های تست (X_test) خروجی‌های پیش‌بینی می‌شوند.

معکوس‌سازی نرمال‌سازی :پیش‌بینی‌ها و مقادیر واقعی که نرمال‌سازی شده بودند، به مقیاس اصلی بازگردانده می‌شوند.

محاسبه معیارها :تابع evaluate_predictions فراخوانی شده و معیارهای ارزیابی محاسبه می‌شوند.

۲ .نحوه اجرای ارزیابی مدل

در این بخش مدل‌ها به صورت **One-Step Prediction** ارزیابی می‌شوند. این روش به این صورت عمل می‌کند که:

- مدل با استفاده از **دنباله‌ای از داده‌های گذشته** به طول seq_length آموزش داده شده و **نقطه بعدی** را پیش‌بینی می‌کند.
- در زمان ارزیابی نیز همین مکانیزم استفاده می‌شود، یعنی به ازای هر دنباله ورودی از داده‌های تست، تنها **یک مقدار** پیش‌بینی می‌شود و سپس دنباله بعدی بررسی می‌شود.

به این ترتیب:

۱. ورودی مدل، seq_lengthداده متوالی گذشته است.

۲. خروجی مدل، پیش‌بینی مقدار یازدهمی در سری زمانی است.

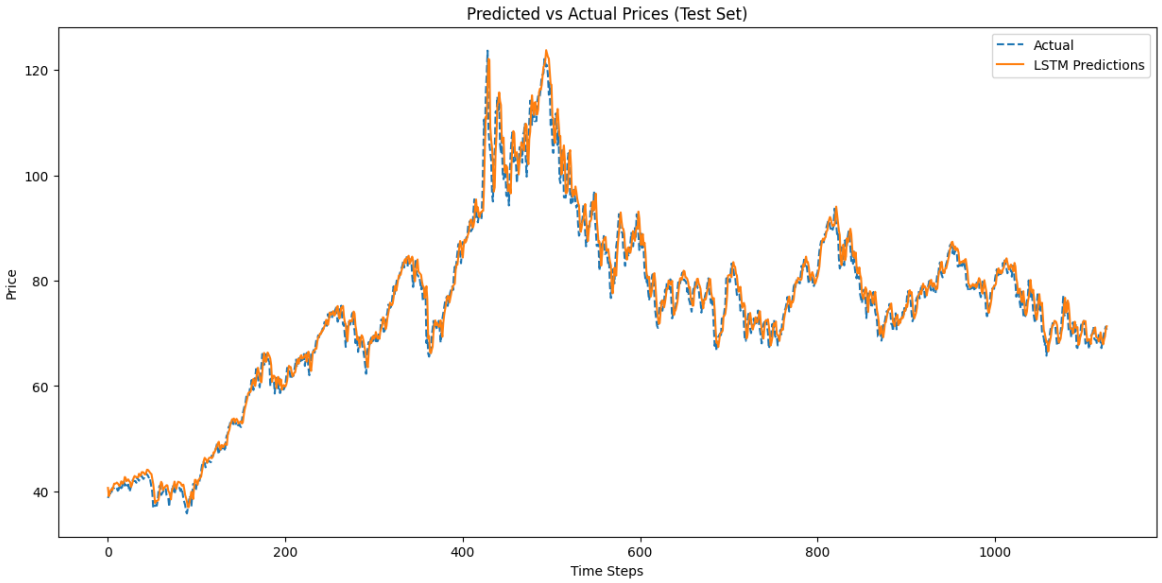
۳. این فرآیند برای تمام داده‌های تست تکرار می‌شود و نتایج پیش‌بینی شده با مقادیر واقعی مقایسه می‌گردند.

این روش ارزیابی به دلیل سادگی و جلوگیری از تجمع خطا در پیش‌بینی‌های بلندمدت انتخاب شده است.

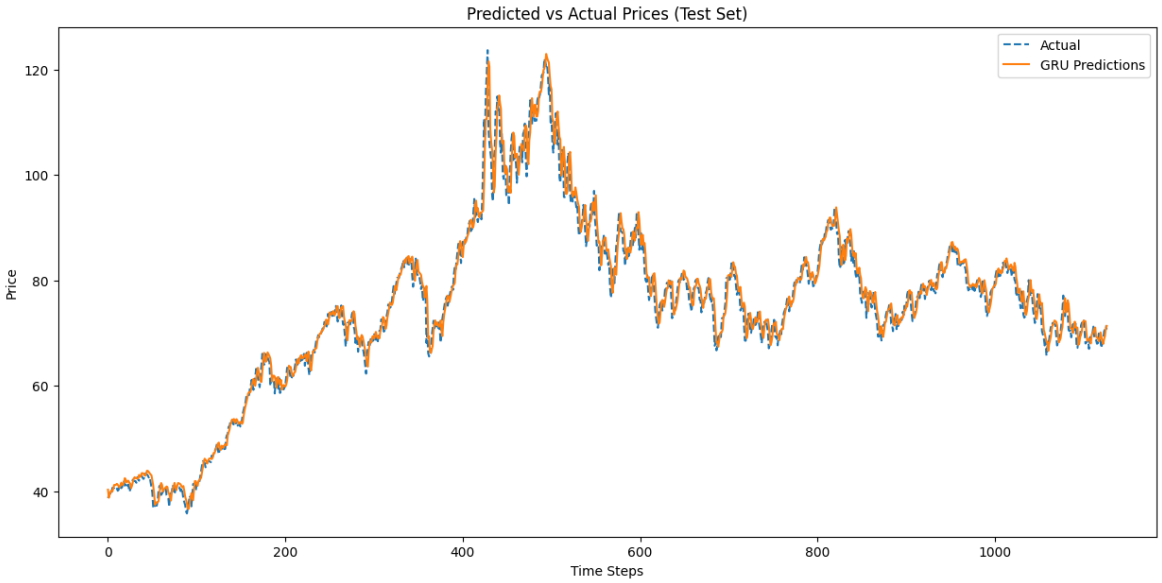
در گام بعد و با توجه به پیشبینی مدل می‌توانیم هر مدل را به صورت جداگانه و در نهایت همه را باهم با واقعیت مقایسه کنیم. برای مقایسه کل مدل‌ها با واقعیت به صورت یک جا کد زیر نوشته شده است. لازم به ذکر است هر مدل هم جداگانه با واقعیت مقایسه شده است.

```
plt.figure(figsize=(15, 7))
plt.plot(actual_inv, label='Actual', linestyle='--')
for model_name, preds in predictions_dict.items():
    plt.plot(preds, label=f'{model_name} Predictions')
plt.title('Predicted vs Actual Prices (Test Set)')
plt.xlabel('Time Steps')
plt.ylabel('Price')
plt.legend()
plt.show()
```

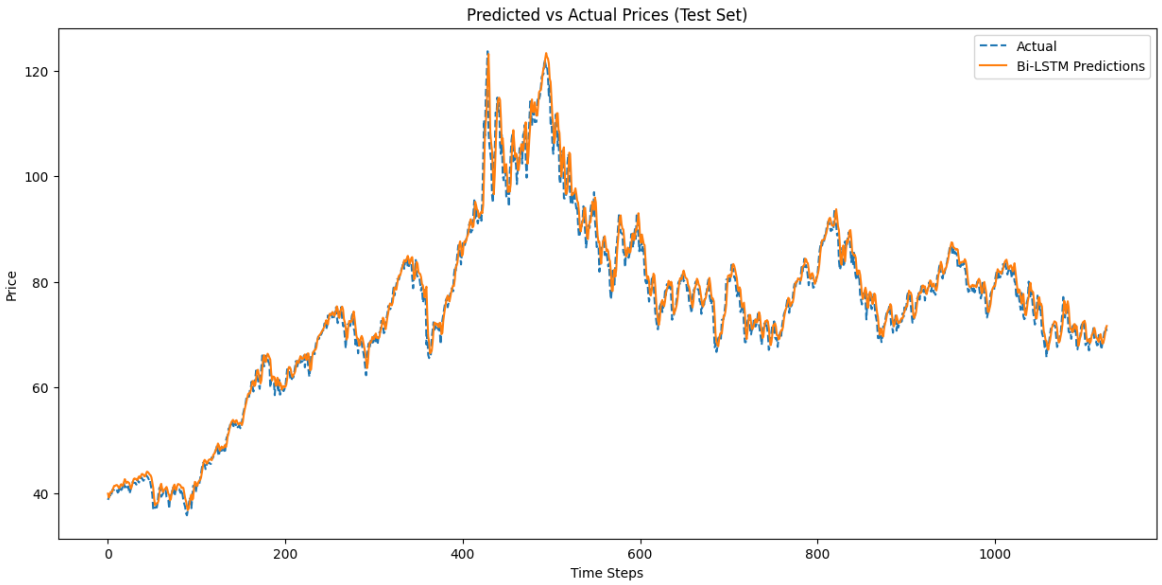
شکل ۲۹ - کد مقایسه مدل‌ها با واقیت



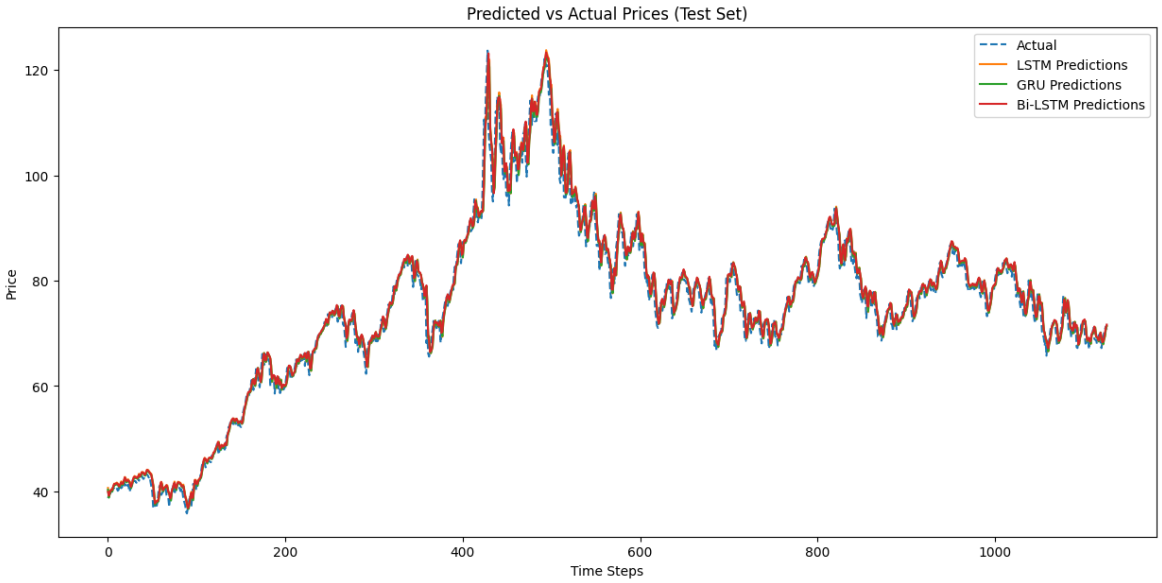
شکل ۳۰- مقایسه مدل **LSTM** با واقعیت



شکل ۳۱- مقایسه مدل **GRU** با واقعیت



شکل ۳۲- مقایسه مدل **Bi-LSTM** با واقعیت



شکل ۳۳- مقایسه تمام مدل‌ها کنار هم با واقعیت

۲-۳-۳. تعریف و مقایسه معیارهای ارزیابی

در این پروژه عملکرد مدل‌ها با استفاده از چهار معیار زیر ارزیابی می‌شود:

۱. میانگین قدر مطلق خطا(MAE)

- **تعریف** :میانگین قدر مطلق خطا، میانگین بزرگی خطاها را بدون در نظر گرفتن جهت آن‌ها محاسبه می‌کند.
- **تفسیر** :هر چه مقدار MAE کمتر باشد، مدل عملکرد بهتری دارد.

$$MAE = (1/n) \sum |y_i - \hat{y}_i|$$

۲. ریشه میانگین مربعات خطا(RMSE)

- **تعریف** RMSE :جذر میانگین مربعات خطا را محاسبه می‌کند و به خطاهای بزرگ حساس‌تر است.
- **تفسیر** :مقدار RMSE پایین‌تر نشان‌دهنده پیش‌بینی‌های دقیق‌تر مدل است.

$$RMSE = \sqrt{[(1/n) \sum (y_i - \hat{y}_i)^2]}$$

۳. میانگین درصد خطای مطلق(MAPE)

- **تعریف** MAPE :خطا را به صورت درصدی از مقادیر واقعی بیان می‌کند و بدون واحد است.
- **تفسیر**:
  - **۱۰٪>** :پیش‌بینی بسیار دقیق.
  - **۲۰٪-۱۰٪** :پیش‌بینی خوب.
  - **۵۰٪>** :پیش‌بینی ضعیف.

$$MAPE = (1/n) \sum |(y_i - \hat{y}_i) / y_i| \times 100$$

۴. ضریب تعیین(R-Squared)

- **تعریف** R^2 :میزان توانایی مدل در توضیح واریانس داده‌های واقعی را نشان می‌دهد.
- **تفسیر**:
  - $R^2 = 1$برازش کامل.
  - $R^2 > 0.8$عملکرد عالی.
  - R^2نزدیک به ۰: عملکرد ضعیف.

$$R^2 = 1 - [\sum (y_i - \hat{y}_i)^2 / \sum (y_i - \bar{y})^2]$$

برای محاسبه این پارامترها تابع زیر نوشته شده است که در فرایند ارزیابی بکار گرفته شد.

```python
def evaluate_predictions(actual, predicted):
    mae = np.mean(np.abs(predicted - actual))
    mse = np.mean((predicted - actual)**2)
    rmse = np.sqrt(mse)
    r2 = 1 - np.sum((actual - predicted)**2) / np.sum((actual - np.mean(actual))**2)
    mape = np.mean(np.abs((actual - predicted) / actual)) * 100
    return {'MAE': mae, 'MSE': mse, 'RMSE': rmse, 'R2': r2, 'MAPE': mape}
```

شکل ۳۴- تابع ارزیابی با معیارهای مورد نظر

جدول ۳- مقایسه و ارزیابی مدل‌ها با **seq_length=10**

LSTM	GRU	Bi-LSTM	
2.419458	1.719686	2.531142	MAE
10.141585	5.820510	10.687362	MSE
3.184586	2.412573	3.269153	RMSE
0.964041	0.979362	0.962105	R2
3.160300	2.248193	3.339909	MAPE (%)

جدول ۴- مقایسه و ارزیابی مدل‌ها با **seq_length=2**

LSTM	GRU	Bi-LSTM	
1.731071	1.679370	1.553836	MAE
6.045829	5.666610	4.849487	MSE
2.458827	2.380464	2.202155	RMSE
0.979058	0.980371	0.983202	R2
2.281083	2.204589	2.052473	MAPE (%)

تحلیل نتایج

۱) MAE

- در نتایج شما با **seq_length = 10**، **Bi-LSTM** کمترین (2.531142) **MAE** را دارد، که نشان‌دهنده دقت کمتر در پیش‌بینی‌ها است.
- در نتایج شما با **seq_length = 2**، **Bi-LSTM** نیز کمترین (1.553836) **MAE** را دارد و دقت بیشتری نسبت به سایر مدل‌ها نشان می‌دهد.
- در مقاله، **GRU** کمترین (1.719686) **MAE** را دارد.

۲) MSE

- در نتایج شما با **seq_length = 10**، **GRU** کمترین (5.820510) **MSE** را دارد.

- در نتایج شما با **seq_length = 2**، **Bi-LSTM** کمترین (4.849487) **MSE** را دارد، که عملکرد بهتری نسبت به مدل‌های دیگر نشان می‌دهد.
- در مقاله، **GRU** کمترین (5.820510) **MSE** را دارد.
- لازم به ذکر است این تابع خطا در فرایند آموزش بود و الزامی به مقایسه در این پارامتر نبود.

۳) RMSE

- در نتایج شما با **seq_length = 10**، **GRU** کمترین (2.412573) **RMSE** را دارد.
- در نتایج شما با **seq_length = 2**، **Bi-LSTM** کمترین (2.202155) **RMSE** را دارد.
- در مقاله، **GRU** کمترین (2.412573) **RMSE** را دارد.

۴) R^2

- در نتایج شما با **seq_length = 10**، **GRU** بیشترین (0.979362) **R^2** را دارد که نشان‌دهنده توضیح بیشتر واریانس داده‌ها توسط این مدل است.
- در نتایج شما با **seq_length = 2**، **Bi-LSTM** بیشترین (0.983202) **R^2** را دارد که به معنای توضیح بیشتر واریانس داده‌ها است.
- در مقاله، **GRU** بهترین (0.979362) **R^2** را دارد.

۵) MAPE

- در نتایج شما با **seq_length = 10**، **GRU** کمترین (2.248193) **MAPE** را دارد.
- در نتایج شما با **seq_length = 2**، **Bi-LSTM** کمترین (2.052473) **MAPE** را دارد.
- در مقاله، **GRU** کمترین (2.248193) **MAPE** را دارد.

نتیجه‌گیری

نتایج با seq_length = 10

- **GRU** در تمامی معیارهاMAE ، MSE، RMSE، R^2وMAPE عملکرد بهتری نسبت به سایر مدل‌ها داشته است و کمترین خطا را در پیش‌بینی‌ها نشان داده است.

نتایج با seq_length = 2

- **Bi-LSTM**در تمامی معیارهاMAE ، MSE، RMSE، R^2و MAPE عملکرد بهتری داشته و کمترین خطا را نسبت به مدل‌های دیگر نشان داده است.

مقایسه با مقاله:

- در مقاله، **GRU**به عنوان بهترین مدل معرفی شده است و در بیشتر معیارها عملکرد بهتری نسبت به **LSTM**و **Bi-LSTM**داشته است.
- در پروژه شما، **Bi-LSTM**با seq_length = 2عملکرد بهتری نسبت به **GRU**و **LSTM**نشان داده است، در حالی که **GRU**در seq_length = 10بهترین عملکرد را دارد.

تفاوت‌ها و علت‌ها:

- **انتخاب seq_length**تأثیر زیادی بر نتایج دارد. با تغییر seq_lengthاز ۱۰ به ۲، **Bi-LSTM** بهتر عمل کرده است[1].
- **Bi-LSTM**توانایی یادگیری الگوهای پیچیده‌تر و بهتر از **GRU**و **LSTM**در این داده‌ها نشان داده است، به خصوص زمانی که طول دنباله کوتاه‌تر است.

۴-۲. ARIMA

۱ .تفاوت مدل‌های ARIMA وSARIMA

مدل(ARIMA (Autoregressive Integrated Moving Average

مدل ARIMA یک مدل آماری است که برای پیش‌بینی داده‌های زمانی به کار می‌رود و سه بخش اصلی دارد:

۱. **Autoregressive (AR)**: قسمت خودبازگشتی که به روابط خطی میان مقادیر گذشته داده‌ها اشاره دارد.

۲. **Integrated (I)**: قسمت یکپارچه‌سازی که نشان‌دهنده تغییرات داده‌ها برای ایستایی (stationarity)است.

[1] الزامی به مقایسه با طول دنباله های متفاوت نبود و به صورت اضافه بر سوال انجام گرفته است.

۳. **Moving Average (MA)**: قسمت میانگین متحرک که به خطاهای پیش‌بینی در داده‌های گذشته مربوط می‌شود.

مدل SARIMA (Seasonal ARIMA)

مدل SARIMA یک مدل پیشرفته‌تر است که به مدل ARIMA برای شبیه‌سازی ویژگی‌های فصلی یا دوره‌ای داده‌ها افزوده می‌شود. مدل SARIMA ویژگی‌های فصلی را نیز در نظر می‌گیرد، که برای داده‌هایی که الگوهای فصلی دارند، مناسب‌تر است.

تفاوت‌های اصلی این دو مدل به شرح زیر است:

- **ARIMA**برای داده‌های غیر فصلی مناسب است.
- **SARIMA**برای داده‌های فصلی طراحی شده است و از چهار پارامتر اضافی برای مدل‌سازی فصول استفاده می‌کندP ، D، Q، Sکه به ترتیب نشان‌دهنده بخش‌های فصلی خودبازگشتی، یکپارچه‌سازی فصلی، میانگین متحرک فصلی، و طول دوره فصلی هستند.

۲ .مزایا و محدودیت‌های مدل ARIMA

مزایا:

- **سادگی و کاربردی بودن** :مدل ARIMA از ساختار ساده‌ای برخوردار است و به راحتی می‌تواند برای پیش‌بینی داده‌های زمانی به کار رود.
- **انعطاف‌پذیری** :می‌توان مدل را برای داده‌های ایستا و غیر ایستا بهینه‌سازی کرد.
- **کاربرد گسترده** ARIMA :به‌طور گسترده در بسیاری از زمینه‌ها مانند پیش‌بینی فروش، تولید، تقاضا، و پیش‌بینی اقتصادی استفاده می‌شود.

محدودیت‌ها:

- **نیاز به ایستایی داده‌ها** :مدل ARIMA نیاز دارد که داده‌ها ایستا باشند، یعنی میانگین و واریانس داده‌ها ثابت باشد. اگر داده‌ها ایستا نباشند، باید فرآیند یکپارچه‌سازی (I) را انجام دهیم.
- **مدل‌سازی فقط داده‌های غیر فصلی** :مدل ARIMA نمی‌تواند ویژگی‌های فصلی داده‌ها را در نظر بگیرد و بنابراین برای داده‌های فصلی ناکارآمد است.
- **عدم شفافیت در شبیه‌سازی روابط پیچیده** ARIMA :قادر به مدل‌سازی روابط پیچیده‌تر و غیرخطی میان داده‌ها نیست.

۳ .مفهوم ریاضی مدلARIMA

مدل ARIMA به صورت ARIMA(p,d,q) مشخص می‌شود که در آن:

- p: تعداد دوره‌های خودبازگشتی.(AR)
- d: تعداد تفاوت‌های لازم برای ایستایی داده‌ها.(I)
- q: تعداد میانگین‌های متحرک.(MA)

فرمول ریاضی مدلARIMA

مدل ARIMA را می‌توان به صورت زیر نوشت:

$$\Phi(B) \times (1 - B)^d \times Y_t = \Theta(B) \times \varepsilon_t$$

که در آن:

- Y_tداده‌های زمانی
- $\Phi(B)$پولینوم خودبازگشتی(AR)
- $\Theta(B)$پولینوم میانگین متحرک(MA)
- Bاپراتور تاخیر
- ε_tخطاهای مدل

توضیح پارامترها:

- **AR(p)**پارامترهای خودبازگشتی که به تأخیرهای گذشته داده‌ها وابسته است.
- **I(d)**فرآیند تفاوت‌گیری که برای ایستایی داده‌ها انجام می‌شود.
- **MA(q)**پارامترهای میانگین متحرک که به خطاهای گذشته بستگی دارند.

۴ .بهینه‌سازی مدل ARIMA و استخراج پارامترهای بهینه

برای بهینه‌سازی مدلARIMA ، باید مقادیر بهینه پارامترهای p، d، و q را تعیین کنیم. برای این کار از روش‌های مختلفی مانند **آزمون خودهمبستگی** ACFو PACF و **جستجوی شبکه‌ای** (Grid Search) استفاده می‌شود.

۴.۱ .تعیین پارامتر

ابتدا باید بررسی کنیم که آیا داده‌ها ایستا هستند یا خیر. اگر داده‌ها ایستا نباشند، باید یک فرآیند تفاوت‌گیری (d) انجام دهیم تا داده‌ها ایستا شوند. به‌طور معمول، از تفاوت اول برای ایستایی استفاده می‌شود.

۴.۲ .تعیین پارامترهای p و q خودبازگشتی و میانگین متحرک

برای انتخاب مقادیر مناسب برای pو qاز نمودارهای **ACF**و **PACF**استفاده می‌شود:

- **PACF**برای انتخاب مقدار pاز نمودار PACF استفاده می‌کنیم که تعداد نقاطی که در آن مقادیر شدید را مشاهده می‌کنیم، معین‌کننده pاست.
- **ACF**برای انتخاب مقدار qاز نمودار ACF استفاده می‌کنیم. تعداد نقاطی که در آن مقادیر شدید مشاهده می‌شود، معین‌کننده qاست.

۴.۳ .پیاده‌سازی در پایتون

برای پیاده‌سازی مدل ARIMA و بهینه‌سازی پارامترها می‌توانیم از کتابخانه‌های **statsmodels**و **pmdarima**استفاده کنیم.

گزارش بهینه‌سازی مدلARIMA

بهینه‌سازی مدل ARIMA با استفاده ازauto_arima

برای بهینه‌سازی مدلARIMA ، از تابع auto_arimaکتابخانه pmdarimaاستفاده کردیم. این تابع به طور خودکار پارامترهای بهینه مدل ARIMA را با بررسی ترکیبات مختلف پارامترهای ppp، ddd، و qqq تعیین می‌کند. مراحل بهینه‌سازی به صورت زیر انجام شد:

۱. **تعریف مجموعه داده آموزشی :**ابتدا داده‌های قیمت بسته‌شده تعدیل‌شده (Adj Close) را از تاریخ ۲۰۱۰-۰۱-۰۱ تا تاریخ جاری دانلود و آماده‌سازی کردیم. سپس ٪۷۰ از داده‌ها را به عنوان مجموعه آموزشی و ٪۳۰ باقی‌مانده را به عنوان مجموعه آزمایشی انتخاب کردیم.

۲. **اجرای :auto_arima** با استفاده از تابع auto_arima، مدل ARIMA را روی داده‌های آموزشی برازش دادیم تا بهترین ترکیب پارامترها مشخص شود. تنظیمات مورد استفاده به شرح زیر بود:

   - **شروع p و :q** از مقدار ۰ شروع می‌شود تا با افزایش تدریجی به حداکثر مقدار ۵ برسد.
   - **تعداد تفاضل‌ها :(d)** یک بار تفاضل‌گیری انجام شد تا داده‌ها ایستا شوند.

- **فصلی نبودن داده‌ها :**به دلیل عدم وجود الگوهای فصلی مشخص در داده‌های نفت خام، گزینه seasonal=Falseانتخاب شد.
- **نمایش جزئیات (trace):** برای کاهش حجم خروجی، trace=Falseتنظیم شد.
- **مدیریت خطاها :**با استفاده از error_action='ignore' و suppress_warnings=True از نمایش خطاهای احتمالی جلوگیری کردیم.
- **روش گام‌به‌گام :**با stepwise=Trueفرآیند جستجو به صورت گام‌به‌گام انجام شد تا سرعت بهینه‌سازی افزایش یابد.

```
# Fit auto_arima on the training data to find optimal (p, d, q)
arima_model_auto = auto_arima(train,
                              start_p=0, start_q=0,
                              max_p=5, max_q=5,
                              d=1,
                              seasonal=False,
                              trace=False,
                              error_action='ignore',
                              suppress_warnings=True,
                              stepwise=True)

# Extract optimal parameters
p, d, q = arima_model_auto.order
arima_order = (p, d, q)
print(f"\nOptimal ARIMA parameters found by auto_arima: p={p}, d={d}, q={q}")
```

شکل ۳۵- بهینه‌سازی پارامترهای مدل **ARIMA**

```
Optimal ARIMA parameters found by auto_arima: p=0, d=1, q=1
```

شکل ۳۶- مقادیر بهینه مدل **ARIMA**

همانطور که در شکل فوق مشخص ازت مقادیر p=0, d=1, q=1 به عنوان مقادیر بهینه مدل گزارش شده اند. این پارامترها نشان‌دهنده این است که مدل ARIMA بدون مؤلفه خودبازگشتی(p=0) ، با یک مرتبه تفاضل‌گیری (d=1) و یک مؤلفه میانگین متحرک (q=1) بهترین عملکرد را در پیش‌بینی داده‌های نفت خام نشان داده است.

۴.۴ .پیاده‌سازی مدل ARIMA و ارزیابی آن

پیاده‌سازیWalk-Forward Validation

روش Walk-Forward Validation یک روش مناسب برای ارزیابی مدل‌های سری زمانی است که به صورت دوره‌ای مدل را با داده‌های جدید به‌روزرسانی می‌کند و پیش‌بینی‌های مرحله به مرحله انجام می‌دهد. این روش به مدل اجازه می‌دهد تا با تغییرات جدید در داده‌ها سازگار شود و عملکرد واقعی‌تری از مدل در پیش‌بینی آینده ارائه دهد.

```python
def walk_forward_validation_arima(train, test, order):
    history = list(train)
    predictions = []

    for actual in test:
        model = ARIMA(history, order=order)
        model_fit = model.fit()
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(actual)

    return predictions
```

شکل ۳۷ - کد روش **walk forward validation**

پیاده‌سازی پیش‌بینی سنتی(Traditional Forecasting)

روش سنتی برازش مدل ARIMA یک بار روی داده‌های آموزشی انجام شده و سپس پیش‌بینی تمام نقاط آزمون به صورت یکجا انجام می‌شود. این روش نسبت به Walk-Forward Validation سریع‌تر است اما انعطاف‌پذیری کمتری دارد و ممکن است با تغییرات جدید در داده‌ها سازگار نباشد. به احتمال قوی مقاله از این روش برای پیشبینی با روش ARIMA استفاده کرده است و ما روش walk forward را به عنوان روش جایگزین با دقت بالا تر اما فقط به صورت روزانه پیشنهاد کرده ایم.

```python
def traditional_arima_forecast(train, test, order):
    model = ARIMA(train, order=order)
    model_fit = model.fit()
    forecast = model_fit.forecast(steps=len(test))
    return forecast
```

شکل ۳۸ - کد ارزیابی نرمال **ARIMA**

در مرحله بعد به سراغ محاسبه پارامترهای مورد سوال برای ارزیابی و اجرای مدل رفتیم و در نهایت نمودار پیشبینی مدل و مقایسه با واقعیت را در هر دو حالت و روش فوق ترسیم کردیم.

```python
def evaluate_predictions(actual, predicted):
    mae = mean_absolute_error(actual, predicted)
    rmse = sqrt(mean_squared_error(actual, predicted))
    mape = np.mean(np.abs((actual - predicted) / (actual + 1e-8))) * 100
    r2 = r2_score(actual, predicted)
    return {'MAE': mae, 'RMSE': rmse, 'MAPE': mape, 'R²': r2}

# Evaluate Walk-Forward Predictions
walk_forward_metrics = evaluate_predictions(test, walk_forward_preds)
print("\nWalk-Forward ARIMA Model Performance:")
for metric, value in walk_forward_metrics.items():
    print(f"{metric}: {value:.4f}")

# Evaluate Traditional Predictions
traditional_metrics = evaluate_predictions(test, traditional_preds)
print("\nTraditional ARIMA Model Performance:")
for metric, value in traditional_metrics.items():
    print(f"{metric}: {value:.4f}")
```

شکل ۳۹ -کد اجرای ارزیابی مدل **ARIMA**

توضیح نحوه کار و بهینه‌سازی مدل

۱. **اجرای Walk-Forward Validation:**

- **هدف :**ارزیابی عملکرد مدل ARIMA در پیش‌بینی داده‌های آینده با به‌روزرسانی مدل در هر گام پیش‌بینی.
- **فرآیند:**
  - شروع با مجموعه داده‌های آموزشی.
  - برای هر نقطه در مجموعه آزمایشی:
    - برازش مدل ARIMA با پارامترهای بهینه روی داده‌های تاریخی فعلی.
    - پیش‌بینی یک گام جلوتر.
    - افزودن مقدار واقعی به تاریخچه برای گام بعدی.
- **مزایا:**
  - ارزیابی واقعی‌تر مدل در شرایط پیش‌بینی واقعی.

- امکان سازگاری مدل با تغییرات جدید در داده‌ها.

۲. **اجرای پیش‌بینی سنتی:**

- **هدف :**روش سنتی برازش مدل ARIMA
- **فرآیند :**برازش مدل ARIMA یک بار روی مجموعه آموزشی و پیش‌بینی تمام نقاط مجموعه آزمایشی به صورت یکجا.

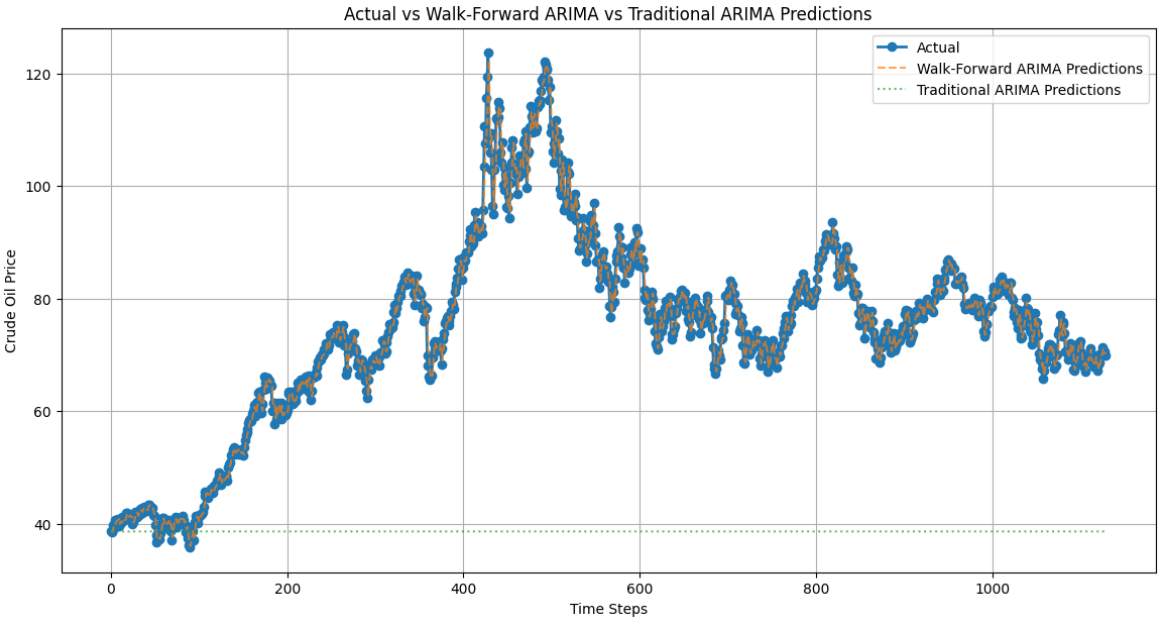
مقایسه روش‌های ارزیابی

در این بخش، دو روش Walk-Forward Validation و پیش‌بینی سنتی ARIMA را با یکدیگر مقایسه می‌کنیم:

ویژگی	Walk-Forward Validation	پیش‌بینی سنتیARIMA
برازش مدل	به‌صورت تکراری در هر گام با تاریخچه جدید	یک بار روی مجموعه آموزشی
پیش‌بینی	پیش‌بینی یک گام به جلو در هر مرحله	پیش‌بینی تمام نقاط مجموعه آزمایشی به صورت یکجا
انعطاف‌پذیری	بالا، مدل می‌تواند با تغییرات جدید سازگار شود	پایین، مدل ثابت می‌ماند و نمی‌تواند با تغییرات جدید سازگار شود
هزینه محاسباتی	بالا، نیاز به برازش مدل در هر گام پیش‌بینی	پایین، نیاز به برازش مدل یک بار
واقع‌گرایی ارزیابی	بالا، شبیه‌سازی شرایط پیش‌بینی واقعی	پایین، ممکن است نتایج با شرایط واقعی تفاوت داشته باشد
پاسخ به تغییرات داده‌ها	مدل می‌تواند با افزودن داده‌های جدید به‌روز شود	مدل نمی‌تواند با داده‌های جدید به‌روز شود

نتیجه‌گیری:

- **Walk-Forward Validation**برای ارزیابی دقیق‌تر و واقع‌گرایانه‌تر مدل‌های سری زمانی توصیه می‌شود، به خصوص زمانی که داده‌ها به طور مداوم در حال تغییر هستند.
- **پیش‌بینی سنتی ARIMA** مناسب برای تحلیل‌های اولیه و زمانی که نیاز به پیش‌بینی سریع و کم‌هزینه است، می‌باشد.



شکل ۴۰- نمودار مقایسه مدل **ARIMA** با واقعیت

نتیجه‌گیری نهایی

در این بخش، با پیاده‌سازی مدل ARIMA و بهینه‌سازی پارامترهای آن با استفاده از auto_arima، عملکرد مدل در دو روش Walk-Forward Validation و پیش‌بینی سنتی ارزیابی شد. نتایج نشان‌دهنده این است که روشWalk-Forward Validation ، ارزیابی دقیق‌تر و انعطاف‌پذیرتری نسبت به روش سنتی ارائه می‌دهد، هرچند هزینه محاسباتی بیشتری دارد. با توجه به ناپایداری و غیرخطی بودن داده‌های قیمت نفت خام، استفاده از روش‌های پیشرفته‌تر و مدل‌های غیرخطی مانند SARIMA یا مدل‌های یادگیری عمیق می‌تواند عملکرد بهتری در پیش‌بینی این نوع داده‌ها ارائه دهد.

جدول ۵- مقایسه و ارزیابی مدل‌ها با **seq_length=10** (مشابه جدول ۶ مقاله)

MAE	RMSE	R2	MAPE (%)	
2.419458	3.184586	0.964041	3.160300	**LSTM**
1.719686	2.412573	0.979362	2.248193	GRU
2.531142	3.269153	0.962105	3.339909	Bi-LSTM
36.0388	39.8066	-4.4970	44.9579	**ARIMA(Traditional)**
1.3960	1.9608	0.9867	27.7217	**ARIMA(Walk forward)**

جدول ۶- مقایسه و ارزیابی مدل‌ها با **seq_length=2** (مشابه جدول ۶ مقاله)

MAE	RMSE	R2	MAPE (%)	
1.731071	2.458827	0.979058	2.281083	**LSTM**
1.679370	2.380464	0.980371	2.204589	GRU
1.553836	2.202155	0.983202	2.052473	Bi-LSTM
36.0388	39.8066	-4.4970	44.9579	**ARIMA(Traditional)**
1.3960	1.9608	0.9867	27.7217	**ARIMA(Walk forward)**

جدول‌های ۳ و ۴ نشان می‌دهد که مدل‌های یادگیری عمیقLSTM ، GRU، Bi-LSTMعملکرد بسیار بهتری نسبت به مدل‌های ARIMA در هر دو روش سنتی و Walk-Forward دارند. به‌ویژه مدل‌های GRUو Bi-LSTM در هر دو جدول دارای کمترین مقادیرMAPE ، RMSEو MAE و بالاترین مقدار R^2هستند که نشان‌دهنده دقت و صحت بالای پیش‌بینی این مدل‌ها می‌باشد.

مدل ARIMA سنتی با مقادیر بسیار بالای خطاها و مقادیر منفی R^2 عملکرد بسیار ضعیفی دارد که نشان‌دهنده عدم توانایی این مدل در پیش‌بینی دقیق قیمت نفت خام است. در مقابل، مدل ARIMA با

روش Walk-Forward بهبود قابل توجهی داشته است اما همچنان در مقایسه با مدل‌های یادگیری عمیق عملکرد ضعیفی از خود نشان می‌دهد.

تحلیل عملکرد مدل‌ها

۱. **مدل‌های یادگیری عمیق:**

- **مزایا:**
  - توانایی بالای مدل‌سازی روابط غیرخطی و پیچیده در داده‌ها.
  - انعطاف‌پذیری در یادگیری الگوهای بلندمدت.
- **نتایج :**مدل‌های GRU و Bi-LSTM در هر دو جدول دارای کمترین خطاها و بالاترین دقت هستند که نشان‌دهنده عملکرد برتر آن‌ها در پیش‌بینی قیمت نفت خام است.

۲. **مدل‌های:ARIMA**

- **مزایا:**
  - سادگی و قابلیت فهم بالا.
  - مناسب برای داده‌های سری زمانی با الگوهای خطی و ایستا.
- **محدودیت‌ها:**
  - عدم توانایی مدل‌سازی روابط غیرخطی و پیچیده.
  - نیاز به ایستا بودن داده‌ها که در صورت ناپایدار بودن، مدل را تحت تأثیر قرار می‌دهد.
- **نتایج :**مدل ARIMA سنتی عملکرد بسیار ضعیفی دارد، اما با استفاده از روش Walk-Forwardبهبود می‌یابد. با این حال، همچنان در مقایسه با مدل‌های یادگیری عمیق دچار کمبودهای قابل توجهی است.

مقایسه با نتایج مقاله

نتایج به‌دست آمده از مدل‌های ARIMA در این تحقیق با نتایج مشابهی که در مقاله ارائه شده‌اند مقایسه شد. جدول‌های ارائه شده در این گزارش مشابه جدول شماره ۶ مقاله می‌باشند که عملکرد مدل‌های مختلف در دو تنظیم مختلف طول توالی seq_length=10 و seq_length=2 را نشان می‌دهند.

مشاهده‌ها:

- **مدل‌های یادگیری عمیق** در هر دو تنظیم طول توالی، به‌ویژه مدل GRU وBi-LSTM ، عملکرد بسیار بهتری نسبت به مدل‌های ARIMA دارند.
- **مدل ARIMA سنتی** دارای خطاهای بسیار بالا و ضریب تعیین منفی است که نشان‌دهنده عدم انطباق مدل با داده‌های غیرخطی و ناپایدار است.
- **مدل ARIMA با Walk-Forward Validation** نسبت به مدل سنتی بهبود یافته است اما همچنان در مقایسه با مدل‌های یادگیری عمیق، عملکرد ضعیفی دارد.

این نتایج با یافته‌های مقاله همخوانی دارد که مدل‌های یادگیری عمیق توانایی بالاتری در پیش‌بینی دقیق قیمت نفت خام نسبت به مدل‌های کلاسیک آماری مانند ARIMA دارند.

نتیجه‌گیری

مدل‌های یادگیری عمیق مانند GRU و Bi-LSTM در پیش‌بینی قیمت نفت خام عملکرد بسیار بهتری نسبت به مدل‌های کلاسیک ARIMA دارند. این امر ناشی از توانایی بالای مدل‌های یادگیری عمیق در مدل‌سازی روابط غیرخطی و پیچیده در داده‌های سری زمانی است. در مقابل، مدل‌های ARIMA به دلیل محدودیت‌های خود در مدل‌سازی الگوهای غیرخطی و نیاز به ایستا بودن داده‌ها، نتوانستند عملکرد قابل قبولی ارائه دهند. با این حال، استفاده از روش Walk-Forward Validation برای مدل ARIMA توانسته است عملکرد مدل را بهبود بخشد، اما همچنان در مقایسه با مدل‌های یادگیری عمیق، نتایج ضعیفی ارائه می‌دهد.