دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر

درس شبکه‌های عصبی و یادگیری عمیق
تمرین چهارم

مبینا مهرآذر - 810100216

محمدرضا محمدهاشمی - 810100206

فهرست

پرسش 1. تشخیص هرزنامه

1-1. مجموعه داده

این مجموعه داده از از سایت Kaggle دریافت کرده‌ایم. این داده شامل ستون‌های text و label می‌باشد که به شرح زیر، 5 سطر اولینه آن نمایش داده شده است:

	text	label
0	من پارسال اصلا آزاد شرکت نک\n.ممنون آقا سامان...	ham
1	بالاخره آزمونارشد تموم شد من\nسلام آقای کریمی...	ham
2	درود بر حاج وحیدی بنده بعنوان یک دکتری تاریخ ...	ham
3	ضمن تقدیر از مسولین محترم\nبا سلام و احترام ...	ham
4	با سلام اینجانب یک دستگاه خودرو پراید 131 با ...	ham

شکل 1. چند سطر از مجموعه داده

همانطور که مشخص است این مجموعه شامل متن‌هایی در ستون text و برچسب متناظر آن در ستون label می‌باشد. این مسئله به مسئله Supervised Classification نیز معروف است که قرار است مدلی آموزش دهیم که برچسب متناظر متن‌های ورودی را با بیشترین دقت و کمترین خطا پیش‌بینی کند. در ادامه به بررسی توزیع این برچسب‌ها می‌پردازیم.

```
df.groupby('label').describe()
```

	text			
	count	unique	top	freq
label				
ham	500	500	مدتی است که همراه اول با پیامک مشتریان خود ر...	1
spam	500	500	سلام به دوستان عزیز\nشما هم میتوانید از این...	1

شکل 2. تعداد داده‌های متناظر با هر برچسب

همانطور که مشخص است، این مجموعه داده شامل 500 داده متعلق به برچسب ham و 500 داده متعلق به برچسب spam می‌باشد.

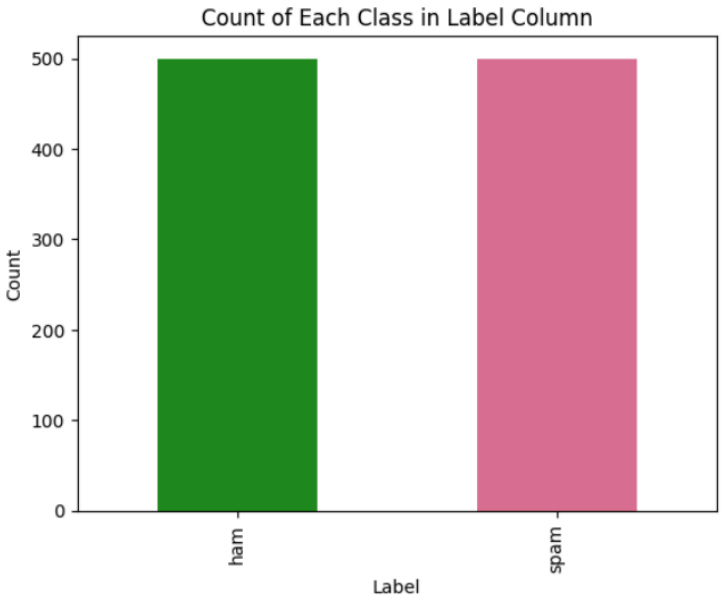
حال به کمک متد info از pandas dataframe، به تحلیل زیر می‌رسیم.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   text    1000 non-null   object
 1   label   1000 non-null   object
dtypes: object(2)
memory usage: 15.8+ KB
```

شکل 3. نتیجه گرفتن info از دیتا فریم

همانطور که از شکل 3 مشخص است، همگی سطر های متناظر در هر ستون، مقادیر غیر تهی دارند. در مجموع 1000 text و 1000 label داریم که از جنس object می‌باشند. میزان حافظه مصرف شده توسط این مجموعه به اندازه $15.8\ KB$ می‌باشد.



شکل 4. نمودار میله‌ای از مجموعه داده

در شکل فوق به بررسی نمودار میله‌ای مجموعه پرداخته‌ایم. همانطور که بررسی شد، تعداد داده‌های متعلق به هر دو برچسب با هم برابر است که نشان از بالانس بودن مجموعه داده ما می‌باشد.

1-2. پیش‌پردازش داده‌ها

برای پیش‌پردازش داده از کتابخانه hazm استفاده کرده‌ایم که در بخش بعد به توضیح آن می‌پردازیم.

1-2-1. کتابخانه hazm

این کتابخانه مخصوص وظایف پردازش زبان طبیعی یا همان NLP[1] مخصوصا برای متون فارسی طراحی شده است که طیف گسترده‌ای از ابزارها و عملکردها را برای تجزیه و تحلیل و پردازش و درک متن فارسی را فراهم می‌آورد.

```
stop_words = set(stopwords_list())

hazm_normalizer = Normalizer()
hazm_stemmer = Stemmer()
hazm_lemmatizer = Lemmatizer()

puncs = [
    '[', '!', '"', '#', '%', '\\', '\"', '(', ')', '*',
    '+', ',', '-', '.', '/', ':', ';', '<', '=', '>', '?', '؟', '@',
    '/', '^', '_', '`', '{', '|', '}', '~',
    '’', '”', '“', '‘', '’', ']','؛','؟','«','»','‘','٪'
]
```

شکل 5. متغیرها و توابع استفاده شده از کتابخانه hazm

ما از توابع زیر استفاده کرده‌ایم:

- Normalizer

این تابع با حذف نشانه‌ها، تصحیح فاصله‌ها، و مدیریت ناسازگاری‌های متن مثل تبدیل اعداد به اعداد معادل در زبان فارسی، متن ورودی را به فرم استاندارد تبدیل می‌کند.

- Stemmer

این تابع بدون درک جامع از معنای یک کلمه، سعی می‌کند ریشه‌ی یک کلمه را با حذف پسوندها و پیشوندهای چسبیده به آن پیدا کند. با این حال اعمال آن روی توکن‌ها کارآمد بوده و مفید است.

- Lemmatizer

[1] Natural Language Processing

این تابع کلمات را بر حسب فهرستی از کلمات مرجع به همراه ریشه‌های مربوطه به شکل پایه یا ریشه‌ی آنها کاهش می‌دهد. این کاهش برای گرفتن نتیجه بهتر در مسائل NLP نتایج دقیق‌تری ارائه می‌دهد و بسیار موثر می‌باشد. ولی در عوض هزینه زمانی بیشتری نسبت به Stemming پرداخت می‌شود.

کلمات Stop Word از لیست stopwords_list از کتابخانه به دست آمده است.

همینطور فهرستی از punctuation های رایج در زبان فارسی تهیه شده است.

1-2-2. پیش پردازش متن فارسی

این مرحله از مراحل ضروری مسائل NLP می‌باشد. در انتهای این مرحله داده‌های ما برای ورود به مدل از لحاظ هماهنگی بین فرمت و حذف شدن نویز موجود آماده می‌شوند.

برای اعمال پیش پردازش روی متن از توابع زیر استفاده شده است:

- **حذف URL ها**: این تابع با کتابخانه re به حذف الگوهای مشابه URL می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

```
def remove_urls(text):
    return re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
```

- **حذف آدرس‌های ایمیل**: این تابع با کتابخانه re به حذف الگوهای مشابه Email می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

```
def remove_emails(text):
    return re.sub(r'\S+@\S+', '', text)
```

- **حذف شماره‌های تلفن**: این تابع با کتابخانه re به حذف الگوهای مشابه شماره‌های تلفن که غالبا 8، 10، یا 11 عددی هستند، می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

```
def remove_phone_numbers(text):
    return re.sub(r'\b\d{8,10,11}\b', '', text)
```

- **حذف تکرار حروف**: این تابع با کتابخانه re به حذف حروف تکرار شوند در هر کلمه می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

```
def remove_repeated_letters(text):
    return re.sub(r'(.)\1+', r'\1', text)
```

- **حذف کلمات توقف**: این تابع کلمات موجود در لیست کلمات توقف را از متن ورودی حذف می‌کند، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

```
def remove_stop_words(words):
    return [word for word in words if word not in stop_words]
```

- **سایر توابع**:

تابع remove_htmls با کتابخانه re به حذف ساختارهای html در متن می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

تابع remove_usernames با کتابخانه re به حذف ساختارهای تگ شده اسم کاربران در متن می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

تابع remove_hashtags با کتابخانه re به حذف ساختارهای هشتگ شده در متن می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

```
def remove_htmls(text):
    return re.sub(re.compile('<.*?>') , '', text)

def remove_usernames(text):
    return re.sub(r'@[^ ]+', '', text)

def remove_hashtags(text):
    return re.sub(r'#', '', text)
```

تابع remove_punctuations با کتابخانه re به حذف punctuation های تعریف شده در کد موجود در متن می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

تابع replace_zwnj_with_space تمامی نیم‌فاصله‌های موجود در متن را به فاصله تبدیل می‌کند. این کار به یکنواختی در متن ورودی مدل و نتیجه بهتر کمک می‌کند.

تابع normalize_text با استفاده از تابع normalizer از کتابخانه hazm، به انجام نرمال‌سازی داده ورودی می‌پردازد.

```
def remove_punctuatoins(text):
    return re.sub(r'[' + re.escape(''.join(puncs)) + r']', ' ', text)

# zwnj : Zero-width non-joiner
def replace_zwnj_with_space(text):
    return text.replace("\u200c", " ")

def normalize_text(text):
    return hazm_normalizer.normalize(text)
```

تابع remove_empty_tokens به حذف توکن‌های خالی استخراج شده از متن می‌پردازد، چرا که عموما معنای خاصی به متن ورودی و spam یا ham بودن آن اضافه نمی‌کنند.

تابع stem_words با استفاده از کتابخانه hazm، به فرایند stemming می‌پردازد.

تابع lemmatize_words با استفاده از کتابخانه hazm، به فرایند lemmatization می‌پردازد.

```
def remove_empty_tokens(words):
    return [word for word in words if word.strip() != '']

def stem_words(words):
    return [hazm_stemmer.stem(word) for word in words]

def lemmatize_words(words):
    return [hazm_lemmatizer.lemmatize(word) for word in words]
```

تابع join_words_into_string تمامی توکن های ورودی را کنار هم قرار داده و یک string از آنها را به عنوان خروجی برمی‌گرداند.تابع stem_words با استفاده از کتابخانه hazm، به فرایند stemming می‌پردازد.

تابع remove_emojies اموجی‌های حاضر در متن را حذف می‌کند. این کاراکترها با کدهای به خصوصی تعریف شده‌اند و در NLP معنای خاصی نمی‌دهند.

تابع remove_bom_characters کاراکترهای از جنس bom را که با کد اسکی مربوطه در متن حاضر هستند را از آن حذف می‌کند.

```python
def join_words_into_string(words):
    return ' '.join(words)

def remove_emojis(text):
    emoji_pattern = re.compile(
        "["
        "\U0001F600-\U0001F64F"  # Emoticons
        "\U0001F300-\U0001F5FF"  # Symbols & pictographs
        "\U0001F680-\U0001F6FF"  # Transport & map symbols
        "\U0001F700-\U0001F77F"  # Alchemical symbols
        "\U0001F780-\U0001F7FF"  # Geometric shapes extended
        "\U0001F800-\U0001F8FF"  # Supplemental arrows
        "\U0001F900-\U0001F9FF"  # Supplemental symbols and pictographs
        "\U0001FA00-\U0001FA6F"  # Chess symbols
        "\U0001FA70-\U0001FAFF"  # Symbols and pictographs extended-A
        "\U00002702-\U000027B0"  # Dingbats
        "\U000024C2-\U0001F251"  # Enclosed characters
        "]+",
        flags=re.UNICODE
    )
    return emoji_pattern.sub(r'', text)

def remove_bom_characters(text):
    # Remove BOM (Byte Order Mark) or any Zero Width No-Break Space
    return text.replace("\uFEFF", "")
```

ترتیب اعمال توابع برای preprocess به شرح زیر است:

ابتدا متن ورودی را که در متغیر text ذخیره شده به توابعی که متن دریافت می‌کنند می‌دهیم. در نهایت فرآیند نرمال‌سازی را اعمال کرده و متن را به توکن هایی استخراج می‌کنیم. سپس توابعی که توکن به عنوان ورودی میگیرند را اعمال کرده و در نهایت تمامی توکن‌های باقی را در یک متغیر از جنس string ذخیره کرده و به عنوان متن preprocess شده خروجی می‌دهیم.

```python
def preprocess_text(text):
    text = remove_urls(text)
    text = remove_emails(text)
    text = remove_phone_numbers(text)
    text = remove_htmls(text)
    text = remove_usernames(text)
    text = remove_hashtags(text)
    text = replace_zwnj_with_space(text)
    text = remove_repeated_letters(text)
    text = remove_punctuatoins(text)
    text = remove_emojis(text)
    text = remove_bom_characters(text)
    text = normalize_text(text)

    words = word_tokenize(text)
    words = remove_empty_tokens(words)
    words = remove_stop_words(words)
    # words = stem_words(words)
    words = lemmatize_words(words)

    text = join_words_into_string(words)
    return text
```

در نهایت متن تمیز شده در هر سطر را در ستون جدید در آن سطر به نام ستون cleaned_text ذخیره کرده و در ادامه از آن استفاده می‌کنیم.

	text	label	cleaned_text
0	من پارسال اصلا آزاد\n.ممنون آقا سامان شرکت نک...	ham	منون آقا سامان پارسال اصلا آزاد شرکت کرد#کن سر...
1	بالاخره آزمونارشد تموم\nسلام آقای کریمی شد من...	ham	سلام کریمی بالاخره آزمونارشد تموم راحت شد#شو ی...
2	درود بر حاج وحیدی بنده بعنوان یک دکتری تاریخ ...	ham	درود حاج وحید بنده بعنوان دکتری تاریخ دستی تار...
3	ضمن تقدیر از مسولین\nبا سلام و احترام محترم ...	ham	سلام احترام تقدیر مسولین محترم سایت تابناک رسا...
4	با سلام اینجانب یک دستگاه خودرو پراید 131 با ...	ham	سلام اینجانب دستگاه خودرو پراید ۱۳۱ شماره درخو...

شکل 6. شمایی از دیتا فریم بعد از preprocess

1-3. نمایش ویژگی

1-3-1. نمایش ویژگی به کمک مدل ParseBERT

مدل استفاده شده در این مسئله یک مدل زبانی از پیش آموزش دیده‌شده[2] می‌باشد که مخصوصا برای متون فارسی طراحی شده است. معماری مدل بر اساس معماری BERT متعلق به Google بوده و بر روی مجموعه بزرگی از متون فارسی Fine-Tune شده است. در این بخش به توضیح مدل می‌پردازیم:

- معماری

این مدل شامل 12 لایه[3]، 768 سلول در لایه نهان[4]، 12 سر توجه[5] می‌باشد. مدل با این معماری نسبتا پیچیده الگوهای پیچیده موجود در متن و وابستگی بین آنها را استخراج می‌کند.

مدل ParseBERT یک مدل برای زبان فارسی است که به منظور استخراج ویژگی‌های متن فارسی مورد استفاده واقع می‌شود.

- **ویژگی Uncased بودن مدل**

این مدل نسبت به حروف بزرگ[6] و کوچک[7] حساس نبوده و در زبانی مثل زبان فارسی که بر خلاف زبان انگلیسی حروف بزرگ و کوچک تعریف نمی‌شوند، ویژگی خوبی می‌باشد.

در ابتدا به کمک نام مدل موارد مورد نیاز را از کتابخانه Hugging Face Transformers دانلود می‌کنیم. config مربوطه، tokenizer مدل و همینطور خود مدل را دانلود می‌کنیم.

- **Tokenizer**

برای تبدیل متن ورودی که غالبا متن‌های خیلی طولانی هستند، به واحدهای کوچک‌تری به نام token در مسائل NLP استفاده می‌شود. در این مسئله از tokenizer مربوط به ParseBERT استفاده کرده‌ایم که از توکن‌سازی WordPiece برای تبدیل متن به زیرکلمه‌ها و در ادامه اندیس‌های عددی استفاده می‌کند. ویژگی‌های این tokenizer به شرح زیر است:

---

[2] Pre-trained language model

[3] Layers

[4] Hidden Layer

[5] Attention Head

[6] Uppercase

[7] Lowercase

1. پیش‌پردازش متن:

این تابع متن ورودی را به واحدهای کوچک‌تر قابل درک برای مدل تقسیم کرده که این token ها می‌توانند کلمه‌ها، تکواژها یا کاراکترهای استخراج شده از متن باشند.

2. حذف نویز موجود در متن:

این تابع می‌تواند علائم نگارشی موجود در متن، نویزهای وارد شده در متن را حذف می‌کند. این مرحله cleaning به مدل کمک می‌کند ساختارهای معنادار را یاد بگیرد و نتیجه بهتری ارائه دهد.

3. مدیریت بردارها:

نیاز است هر توکن از متن فارسی برای ورود به مدل به برداری از اعداد تبدیل شود تا مدل قابلیت پردازش روی آن را داشته باشد. با تبدیل درست همه‌ی توکن‌ها به فضای برداری و رعایت ارتباط بین آنها در تبدیل صورت گرفته، مدل روابط معنایی را بهتر درک خواهد کرد.

4. مدیریت token های خاص:

در استخراج توکن از متن نیاز است یک سری توکن خاص در بین توکن‌های استخراج شده جایگذاری شوند. مانند CLS که مخصوص شروع جمله و توکن SEP که مخصوص جداسازی جملات از همدیگر است.

5. پشتیبانی از تکواژها در متن:

این تابع به خوبی یاد گرفته کلمات زبان فارسی را به تکواژهای تشکیل دهنده‌ی آن بشکند که مدلی که آن‌ها را به عنوان ورودی دریافت می‌کند، به خوبی روابط معنایی را بیاموزد و به دقت خوبی برسد.

حاشیه گذاری :

این روش یک روش جهت یکسان سازی طول ورودی به هدف تسهیل پردازش داده ورودی است که در آن برای داده های به قدر کافی بلند کاری نکرده و برای داده های کوتاه فضای باقی مانده را با فضای خالی پر میکنیم.

1-3-3. ویژگی‌های بردار تعبیه

ابعاد پیش‌فرض بردار تعبیه یا Embedding Vector در ParsBERT، به اندازه 768 می‌باشد. این ابعاد به اندازه تعداد ویژگی‌هایی است که هر توکن به وکتور عددی تبدیل می‌شود. به طور کلی ابعاد بردار تعبیه نشان‌دهنده‌ی تعداد ویژگی‌هایی است که مدل مربوطه برای نمایش هر توکن از

متن استفاده می‌کند. در کدام از این تعداد مربوط به یک ویژگی به خصوصی است مانند معنی توکن مربوطه ، نقش دستوری توکن مربوطه در کل متن، ارتباط معنای توکن مربوطه با بقیه توکن‌ها، موقعیت توکن مربوطه در جمله‌ای که به آن تعلق دارد، و یا ویژگی‌های دیگر در NLP. اندازه این بردار رابطه مستقیمی با میزان پیچیدگی آن دارد.

در مدل ParsBERT، بردارهای تعبیه حاصل از ترکیب سه مولفه تشکیل می‌شوند.

- نمایش معنای اولیه token یا Token Embeddings
- اطلاعات مربوط به موقعیت token در جمله یا Positional Embeddings
- تمایز میان بخش‌های مختلف جمله یا Segment Embeddings

در واقع ما در ساخت بردار تعبیه word embedding انجام می‌دهیم. به این معنی که ویژگی‌های معنایی و نحوی کلمه‌های تجزیه‌شده را به فضای چند بعدی جدید منتقل می‌کند. این فرآیند ارتباط معنایی بین توکن‌های مختلف در فضای برداری جدید نیز حفظ می‌شود. به این معنی که اگر دو کلمه در زبان فارسی معنای خیلی نزدیک به هم داشته باشند، تبدیل یافته آن دو کلمه در فضای جدید نیز به همدیگر نزدیک می‌مانند این اصل رعایت شده برای بقیه ارتباط‌های معنایی نیز صادق است. در نتیجه تبدیل صورت گرفته مدل قادر است با استفاده از بردارهای حاصل مسئله طبقه‌بندی متن را حل کند. برای مثال، کلماتی مثل 'پدر' و 'بابا' که هم معنی هستند، در فضای برداری جدیدی نزدیک به هم قرار می‌گیرند. همینطور 'پدر' و 'خانواده' نیز نزدیک به هم خواهند بود.

1-4. ساخت مدل

1-4-1. تجزیه مجموعه داده

مجموعه داده را به کمک تابع split به سه دسته train، validation، و test با نسبت‌های 56%، 14%، و 30 درصد تقسیم کرده‌ایم.

ابعاد داده‌ها به صورت زیر قابل مشاهده می‌باشد:

```
Dimensions of the datasets:
x_train: (560, 32, 120)
x_test:  (300, 32, 120)
y_train: (560, 2)
y_test:  (300, 2)
x_val:   (140, 32, 120)
y_val:   (140, 2)
```

شکل 7. ابعاد داده‌های ترین و تست و ولیدیشن

همانطور که مشخص است، همه‌ی دسته‌های y که برچسب داده می‌باشند، ابعاد به صورت (d_1, d_2) دارند که در آن d_1 تعداد سطرهای داده و d_2 تعداد لیبل‌ها به صورت spam و ham می‌باشد. همینطور همه‌ی دسته‌های x که داده ورودی هستند، ابعاد به صورت (d_1, d_2, d_3) دارند که در آن d_3 اندازه کاهش یافته‌ی بردار تعبیه می‌باشد. d_2 نیز طول جملات است که با padding یا حاشیه‌گذاری به اندازه 32 تنظیم شده است. d_1 تعداد سطرهای داده می‌باشد.

1-4-2. مدل CNN-LSTM

```python
def compile_model(model, optimizer, learning_rate):
    model.compile(
        loss='categorical_crossentropy',
        optimizer=optimizer(learning_rate=learning_rate),
        metrics=['accuracy']
    )
    return model
```

```python
def build_cnn_lstm(batch_size, learning_rate, optimizer):
    input_layer = Input(shape=(32, 120))
    cnn = Conv1D(filters=32, kernel_size=3, activation='relu', padding='same')(input_layer)
    cnn = MaxPooling1D(pool_size=2)(cnn)  # Reduces timesteps by half, features stay the same
    lstm = LSTM(64, return_sequences=False, dropout=0.2)(cnn)
    output = Dense(2, activation='softmax')(lstm)

    model = Model(inputs=input_layer, outputs=output)
    return compile_model(model, optimizer, learning_rate)
```

همانطور که قابل مشاهده ابتدا یک لایه کانولوشنی با اکتیویشن رلو داریم که خروجی آن را به یک مکس پولینگ داده و خروجی مکس پولینگ را وارد lstm میگنم و سپس به کمک یک لایه فولی کانکتد با اکتیویشن سافت مکس مدل CNN-LSTM را به کمک کتابخوانه کراس کامپایل کرده و ریترن میکنیم.

1-4-3. مدل CNN

```python
def compile_model(model, optimizer, learning_rate):
    model.compile(
        loss='categorical_crossentropy',
        optimizer=optimizer(learning_rate=learning_rate),
        metrics=['accuracy']
    )
    return model
```

```
def build_simple_cnn(batch_size, learning_rate, optimizer):
    input_layer = Input(shape=(32, 120))
    cnn = Conv1D(filters=32, kernel_size=3, activation='relu', padding='same')(input_layer)
    cnn = MaxPooling1D(pool_size=2)(cnn)
    cnn = Flatten()(cnn)
    output = Dense(2, activation='softmax')(cnn)

    model = Model(inputs=input_layer, outputs=output)
    return compile_model(model, optimizer, learning_rate)
```

در این مدل نیز همانطور که قابل مشاهده است یک لایه کانولوشنی را به مکس پول متصل کرده و خروجی را تک بعدی کرده و به کمک یک لایه فولی کانکتد با اکتیویشن فانکشن سافت مکس پردیکشن را انجام میدهیم و تمام مدل را به کمک کراس کامپایل میکنیم .

1-4-4. مدل LSTM

```
def compile_model(model, optimizer, learning_rate):
    model.compile(
        loss='categorical_crossentropy',
        optimizer=optimizer(learning_rate=learning_rate),
        metrics=['accuracy']
    )
    return model
```

```
def build_simple_lstm(batch_size, learning_rate, optimizer):
    input_layer = Input(shape=(32, 120))
    lstm = LSTM(64, return_sequences=False, dropout=0.2)(input_layer)
    output = Dense(2, activation='softmax')(lstm)

    model = Model(inputs=input_layer, outputs=output)
    return compile_model(model, optimizer, learning_rate)
```

در این مدل نیز ورودی را به یک مدلLSTM میدهیم و خروجی را به شبکه فولی کانکتد میدهیم و مدل را کامپایل میکنیم.

1-4-5. جست و جوی حریصانه

برای پیدا کردن بهترین هایپر پارامترهای مدل از الگوریتم جست و جوی حریصانه استفاده شده است. بهترین هایپر پارامترهای هر مدل به شرح جدول زیر می‌باشد:

جدول 1. هایپر پارامترهای سه مدل

Model	Batch Size	Learning Rate	Optimizer

CNN-LSTM	**8**	**0.0001**	**Adam**
CNN	**8**	**0.001**	**Adam**
LSTM	**8**	**0.001**	**Adam**

1-4-6. نقاط قوت و ضعف مدل‌های CNN

مدل CNN یا همان Convolutional Neural Network عملکرد خیلی خوبی در استخراج ویژگی‌های مکانی و فضایی از روی داده ورودی مانند جنس بافت شیء، لبه‌های شیء دارد. این عملکرد به علت ماهیت اعمال فیلتر روی داده ورودی می‌باشد. همینطور در این مدل از Pooling نیز استفاده می‌شود که در نتیجه‌ی آن تعداد پارامترها کاهش می‌یابند. تعادل در اندازه تعداد پارامترهای مدل در هزینه‌ای که بابت زمان و منابع می‌پردازیم بسیار موثر می‌باشد. همینطور لایه های کانولوشن پارامترها را به اشتراک می گذارند که تعداد پارامترها و هزینه محاسباتی را کاهش می دهد. لایه های کانولوشن پارامترها را به اشتراک می گذارند که تعداد پارامترها و هزینه محاسباتی را کاهش می دهد. همینطور CNN ها می توانند الگوها را بدون توجه به موقعیت آنها در داده های ورودی تشخیص دهند.

با این حال معایب CNN ها به این شرح است که در این مدل‌ها معمولا ورودی‌ها با اندازه‌های ثابت و فیکس به مدل داده می‌شوند. معماری آنها ذاتا برای مدیریت داده‌های متوالی یا زمانی طراحی نشده است.

1-4-7. نقاط قوت و ضعف مدل‌های LSTM

مدل LSTM یا همان Long Short-term Memory Network عملکرد خیلی خوبی در داده‌های ترتیبی یا زمانی ماند text ،voice، و سری‌های زمانی دارد. این مدل با سلول‌های حافظه[8] موجود در معماری‌اش می‌تواند اطلاعات را برای دوره‌های طولانی ذخیره کند و مشکل ناپدید شدن

[8] Memory Cells

گرادیان[9] را برطرف کند. همینطور مدل از گیت‌های ورودی[10]، خروجی[11]، و فراموشی[12] برای کنترل جریان اطلاعات و حفظ وابستگی‌های طولانی مدت استفاده می‌کند.

از مزایای مدل می‌توان به انعطاف‌پذیری مدل در سایز ورودی اشاره کرد. همینطور مدل LSTM برای ثبت وابستگی‌های طولانی مدت در داده‌های متوالی طراحی شده و برای پردازش سری‌های زمانی و زبان طبیعی بسیار مناسب می‌باشد.

از معایب مدل پیچیدگی محاسباتی مدل می‌توان نام برد که در مقایسه با مدل‌های ساده‌تر، هزینه زمانی و منابع بیشتری دارد.

1-4-8. هدف از ادغام دو مدل

با ادغام این دو مدل در واقع از نقاط قوت دو معماری برای ایجاد یک مدل ترکیبی قدرتمند استفاده می‌کنیم. با این رویکرد روی داده‌هایی که شامل ویژگی‌های مکانی و زمانی هستند، مثل تجزیه و تحلیل ویدیو، تشخیص گفتار و پردازش زبان طبیعی (مانند مسئله تشخیص هرزنامه) کاربرد خوبی دارد.

1-5. ارزیابی

متریک‌های مقایسه در مقاله برای 3 مدل به شرح جدول زیر است.

	Model	Accuracy	Precision	Recall	F1-Score	AUC
0	CNN-LSTM	0.940000	0.940078	0.940000	0.939997	0.976089
1	Simple-CNN	0.936667	0.936686	0.936667	0.936666	0.984400
2	Simple-LSTM	0.913333	0.913996	0.913333	0.913299	0.963778

شکل 8. نتایج سه مدل روی متریک‌های تعریف‌شده

همانطور که مشاهده می‌شود، مدل CNN-LSTM در تمامی متریک‌ها به جز AUC به نتیجه بهتری رسیده ولی با این حال، مقادیر بسیار به هم نزدیک می‌باشند. به خصوص دو مدل CNN و CNN-LSTM به همدیگر نزدیکتر هستند و دلیل آن، ماهیت مجموعه داده و مسئله مورد بررسی می‌باشد.

---

[9] Vanishing Gradient Problem

[10] Input Gates

[11] Output Gates

[12] Forget Gates

1-6. امتیازی

1-6-1. استفاده از روش کیسه‌ی کلمات برای نمایش ویژگی

در این بخش به کمک روش کیسه‌ی کلمات[13] برای هر جمله از متن مربوطه یک بردار ساخته می‌شود. مدل کیسه‌ی کلمات یا BoW یک تکنیک اساسی و پرکاربرد در پردازش زبان طبیعی (NLP) و متن کاوی است. این مدل یک روش ساده و موثر برای تبدیل متن به نمایش‌های عددی ارائه می‌دهد. این روش در عین سادگی محدودیت‌هایی مانند اهمیت نداشتن ترتیب کلمات در متن و عدم اهمیت به معنای کلمات دارد که برای حل این مشکلات مدل جدیدتر TF-IDF ارائه شده است.

Text Representation: مدل BoW متن را به عنوان مجموعه ای از کلمات نشان می دهد که دستور زبان و ترتیب کلمات را نادیده می گیرد اما تعدد را حفظ می کند. هر متن ورودی به عنوان بردار تعداد کلمات نمایش داده می شود.

Vocabulary Creation: مدل واژگانی از تمام کلمات منحصر به فرد موجود در مجموعه ایجاد می کند. به هر کلمه در واژگان یک شاخص منحصر به فرد اختصاص داده شده است.

Vectorization: هر متن ورودی به یک بردار با طول ثابت تبدیل می شود که در آن هر عنصر تعداد یک کلمه خاص در متن ورودی را نشان می دهد. بنابراین طول این بردار با تعداد کلمات واژگان متن برابری می‌کند و مقادیر آن‌ها نشان‌دهنده‌ی فراوانی هر کلمه در متن است.

```
vectorizer = CountVectorizer(max_features=120)
X_BoW = vectorizer.fit_transform(df['cleaned_text']).toarray()
X_train_BoW, X_test_BoW, y_train_BoW, y_test_BoW = split(X_BoW, labels, ratio=0.3)
```

تابع CountVectorizer از کتابخانه sklearn استفاده شده که تعداد max_features آن به مقدار 120 محدود شده تا شرایط مشابهی برای همگی مدل‌ها فراهم شود. همینطور مجموعه داده به نسبت 70 به 30 برای آموزش و تست تقسیم شده است.

1-6-2. آموزش روی چندین مدل سنتی یادگیری ماشین

مدل‌های استفاده شده در این بخش به شرح زیر هستند:

[13] Bag of Words

```python
traditional_models = {
    "Support Vector Machine": SVC(probability=True, kernel="linear", random_state=42),
    "AdaBoost": AdaBoostClassifier(n_estimators=50, random_state=42),
    "Bagging Classifier": BaggingClassifier(n_estimators=50, random_state=42),
    "Extra Trees": ExtraTreesClassifier(n_estimators=50, random_state=42),
    "Logistic Regression": LogisticRegression(random_state=42, max_iter=1000),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "Multinomial Naive Bayes": MultinomialNB(),
}
```

.

3-6-1. مدل‌های سنتی یادگیری ماشین

برای این بخش به توضیح مدل‌های ارائه شده می‌پردازیم.

مدل **Support Vector Machine** یا SVM مدل supervised learning قدرتمندی برای مسائل طبقه‌بندی می‌باشد. آنها با یافتن ابرصفحه بهینه[14] کار می کنند که نقاط داده کلاس های مختلف را با حداکثر حاشیه[15] جدا می کند. SVM ها در فضاهای با ابعاد بالا و به ویژه برای مسائل طبقه بندی باینری مفید هستند. این مدل‌ها می‌توانند داده‌های خطی و غیر خطی را با استفاده از توابع هسته‌ای[16] مانند توابع چند جمله‌ای، توابع پایه شعاعی[17] و هسته‌های سیگموید[18] مدیریت می‌کنند. این مدل‌ها در برابر overfitting مخصوصا در فضاهای ویژگی با ابعاد بالا مقاوم هستند.

مدل **AdaBoost** یا همان Adaptive Boosting، یک تکنیک ensemble learning است که چندین مدل طبقه‌بند ضعیف را برای ایجاد یک مدل طبقه‌بند قوی ترکیب می‌کند. مدل این عمل را با آموزش مکرر طبقه‌بندی‌کننده‌های ضعیف‌تر که معمولاً درخت‌های تصمیم‌گیری هستند، انجام می‌دهد و وزن‌های آن‌ها بر اساس عملکردشان تنظیم می‌کند. در این روش به نمونه‌هایی که به اشتباه طبقه‌بندی شده اند وزن‌های بیشتری اختصاص داده می‌شود تا طبقه‌بندی‌کننده‌های بعدی روی آنها بیشتر تمرکز کنند. در واقع مدل از چندین classifier ضعیف‌تر استفاده کرده و با کنار هم قرار دادن آنها دقت نهایی را بالا بردن و خطر overfitting را کاهش می‌دهد.

مدل **Bagging Classifier** یا Bootstrap Aggregating، یک روش یادگیری گروهی[19] است که هدف آن بهبود پایداری و دقت مدل‌های یادگیری ماشین است. روش مدل این است که چندین

---

[14] Optimal Hyperplane

[15] Margin

[16] Kernel Functions

[17] Radial basis function (RBF)

[18] Sigmoid Kernel

[19] Ensemble learning

نمونه از یک مدل را بر روی زیرمجموعه‌های مختلفی از داده که توسط فرآیند bootstrapping[20] حاصل شده، آموزش می‌دهد. پیشبینی‌های این مدل‌ها معمولا با روش‌هایی مانند میانگین‌گیری برای رگرسیون یا گرفتن رای اکثریت تجمیع می‌شوند. روش Bagging در کاهش واریانس و جلوگیری از overfitting عملکرد خیلی خوبی دارد. مدل‌های استفاده شده هم اکثرا درخت‌های تصمیم هستند که در نتیجه مدلی مانند جنگل تصادفی می‌دهد.

مدل **Logistic Regression** یک مدل آماری است که در مسائل طبقه‌بندی باینری استفاده می‌شود. این مدل احتمال یک نتیجه باینری را بر اساس یک یا چند متغیر پیشبینی به کمک تابع logistic به دست می‌آورد. این مدل ضرایب متغیرهای پیش بینی را از طریق تخمین حداکثر درست‌نمایی[21] تخمین می زند. ویژگی این مدل سادگی و قابل تفسیر بودن آن است. از این مدل می‌توان برای حل مسائل طبقه‌بندی چند کلاسه از طریق تکنیک‌هایی مانند one-vs-rest و softmax استفاده کرد.

1-6-4. ارزیابی روی مدل‌های سنتی یادگیری ماشین

در جدول زیر به بررسی نتایج می‌پردازیم:

	Model	Accuracy	Precision	Recall	F1-Score	AUC
0	CNN-LSTM	0.940000	0.940078	0.940000	0.939997	0.976089
1	Simple-CNN	0.936667	0.936686	0.936667	0.936666	0.984400
2	Simple-LSTM	0.913333	0.913996	0.913333	0.913299	0.963778
3	Support Vector Machine	0.920000	0.921198	0.920000	0.919943	0.972933
4	AdaBoost	0.920000	0.920075	0.920000	0.919996	0.965800
5	Bagging Classifier	0.930000	0.930938	0.930000	0.929962	0.964489
6	Extra Trees	**0.950000**	**0.950180**	**0.950000**	**0.949995**	0.980889
7	Logistic Regression	0.936667	0.936686	0.936667	0.936666	0.982756
8	Random Forest	0.926667	0.926970	0.926667	0.926654	**0.985111**
9	Multinomial Naive Bayes	0.913333	0.914512	0.913333	0.913272	0.976467

شکل 9. نتایج همه‌ی مدل‌ها روی متریک‌های تعریف‌شده

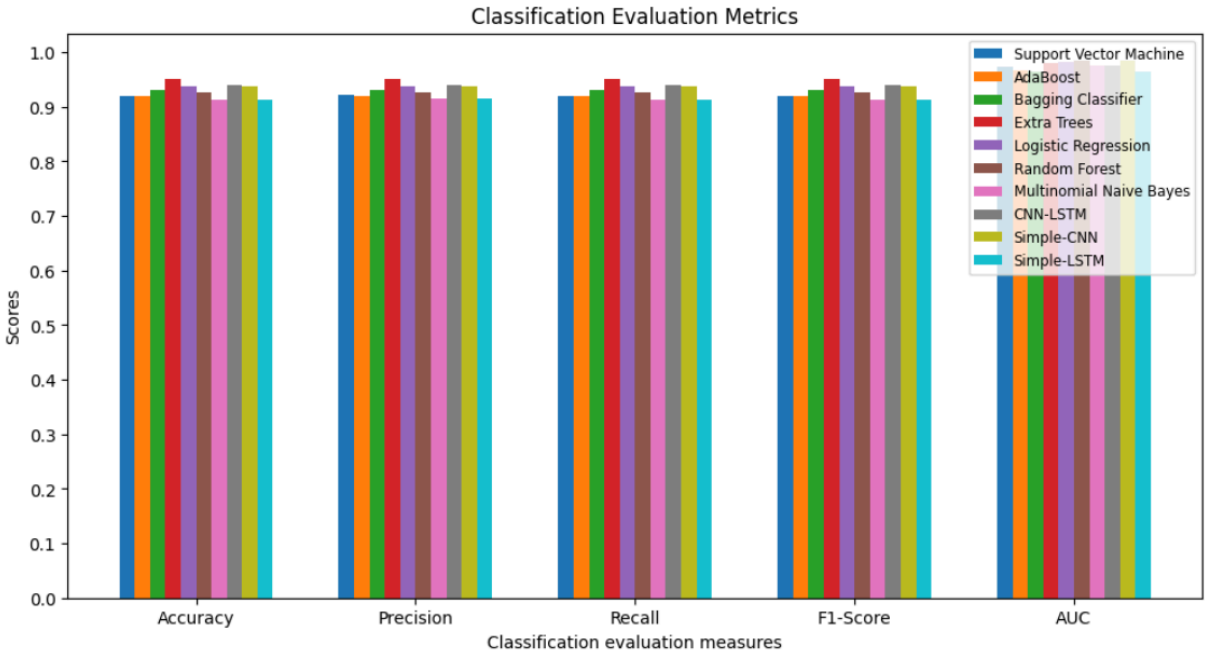
[20] Random Sampling with Replacement

[21] maximum likelihood estimation

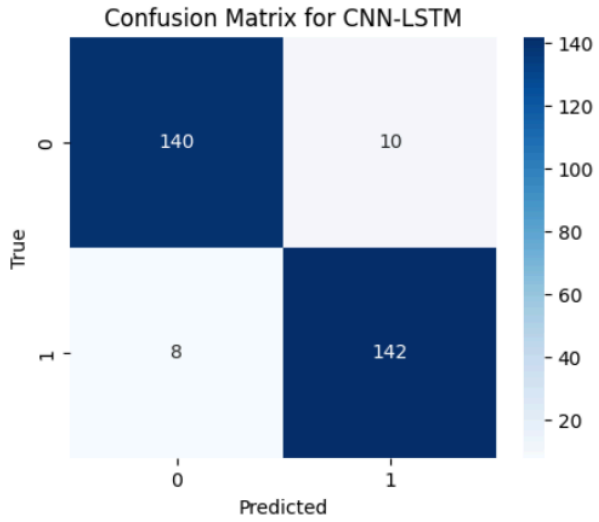
نتایج نشان می‌دهند که مدل‌های کلاسیک یادگیری ماشین مانند **Extra Trees** و **Logistic Regression** عملکرد بهتری نسبت به مدل‌های LSTM ساده داشته‌اند. دلیل این موضوع می‌تواند شامل موارد زیر باشد:

قدرت مدل های دیپ لرنینگ به تشخیص الگو ها و ایجاد پترن های ارزشمند بر اساس داده های ورودی است که در واقع بخشی از فرایند فیچر اینجینیرینگ توسط همین مدل ها به عهده گرفته شود ولی در تسک ما از آنجا که هم در مدل های آماری و هم در مدل های دیپ لرنینگ از امبدینگ های حاصل از برت استفاده شده که فیچر های بسیار با کیفیتی هستند مدل های کلاسیک را قادر به نشان دادن عملکردی بسیار خوب مینماید و همچنین به دلیل ابعاد کوچک دیتا ست امکان اور فیت در مدل های دیپ لرنینگ و یا خوب ترین نشدن مدل بالا تر است.

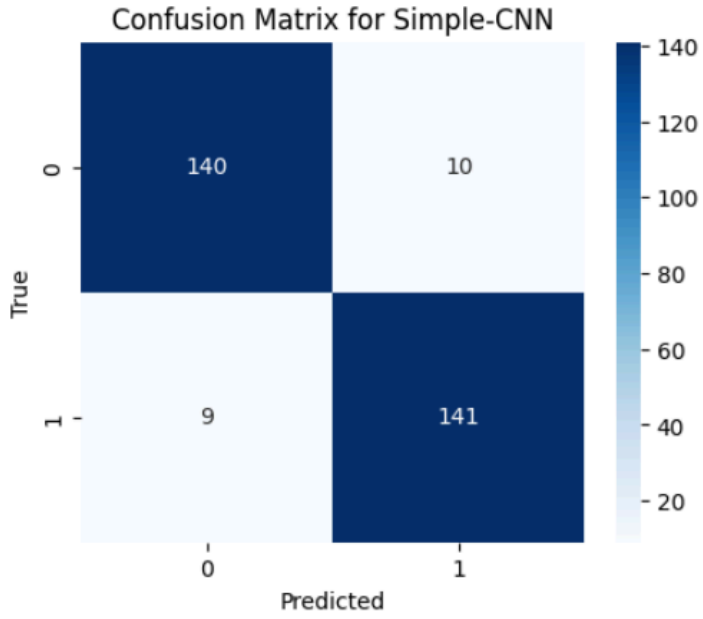
در ادامه چندین نمودار و ماتریس آشفتگی برای درک بهتر نتایج مدل‌های مورد بررسی ارائه شده است.



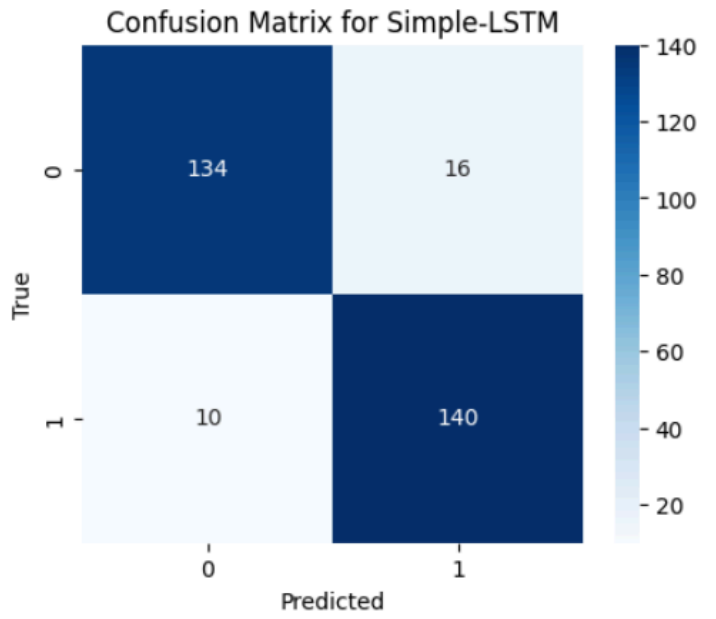
شکل 10. نمودار میله‌ای متریک‌ها



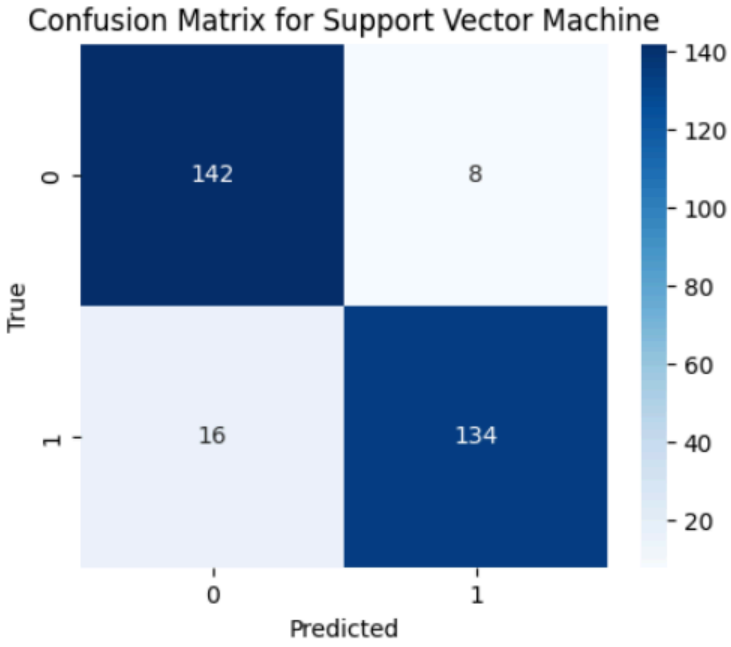
شکل 11. ماتریس سردرگمی مدل CNN-LSTM



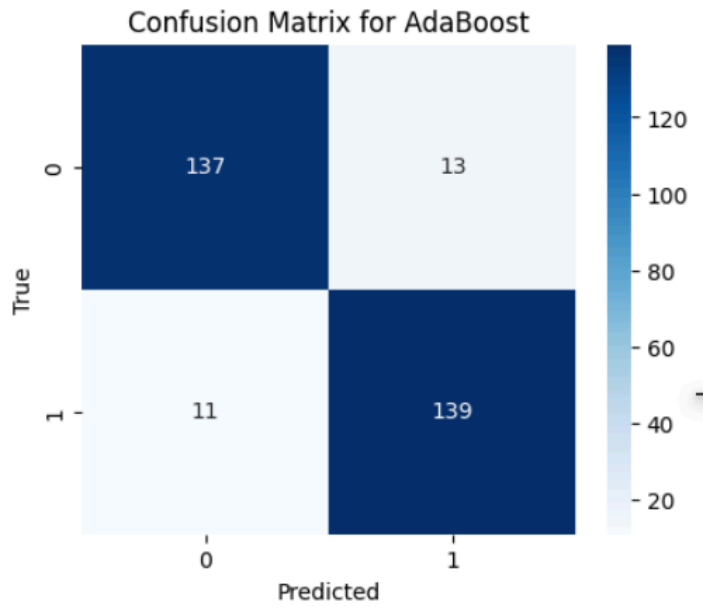
شکل 12. ماتریس سردرگمی مدل CNN



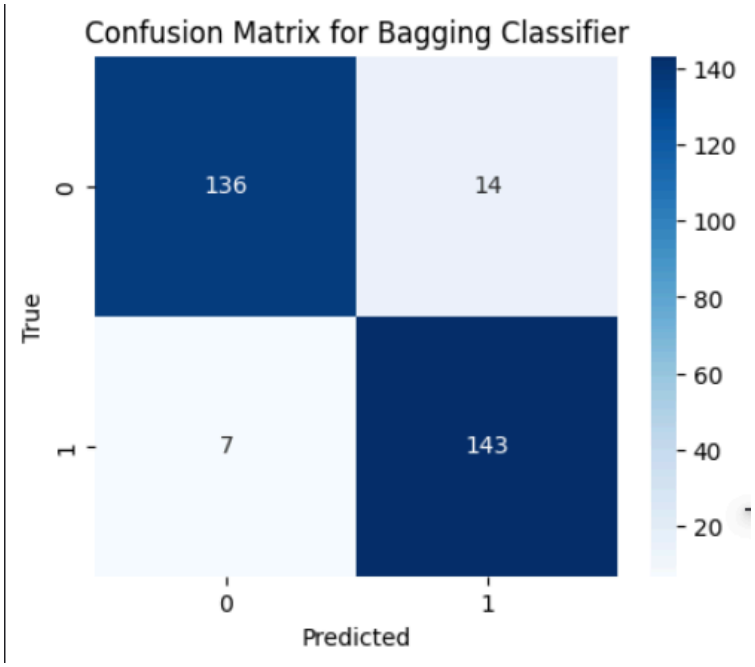
شکل 13. ماتریس سردرگمی مدل LSTM



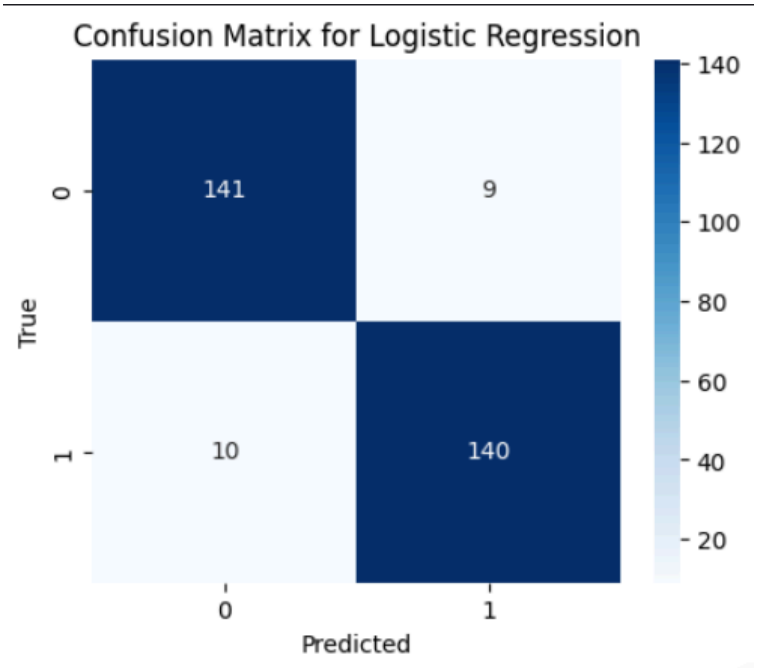
شکل 14. ماتریس سردرگمی مدل SVM



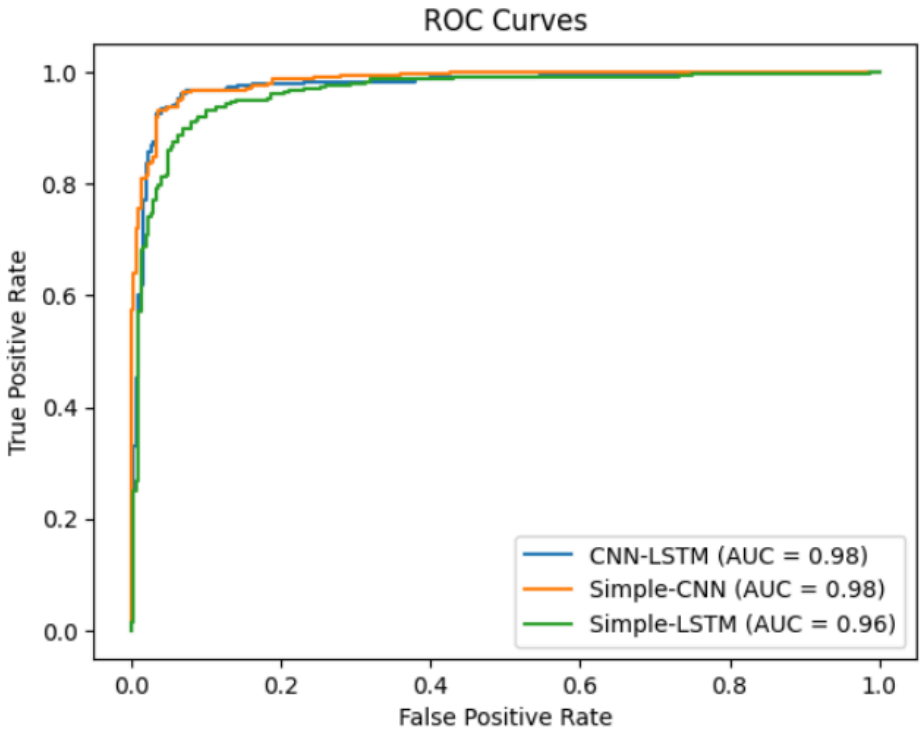
شکل 15. ماتریس سردرگمی مدل AdaBoost



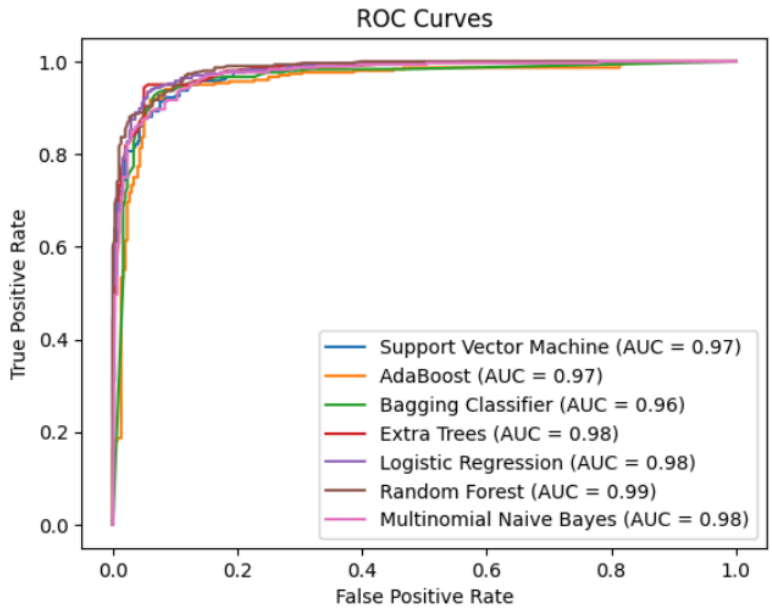
شکل 16. ماتریس سردرگمی مدل Bagging Classifier



شکل 17. ماتریس سردرگمی مدل Logistic Regression



شکل 18. نمودار ROC سه مدل CNN, LSTM, SNN-LSTM



شکل 19. نمودار ROC چهار مدل ML

پرسش 2 - پیش‌بینی ارزش نفت

2-1. مقدمه

2-2. مجموعه دادگان و آماده‌سازی

2-2-1. دانلود و معرفی مجموعه داده

در این مسئله از کتابخانه yfinance برای واکشی داده‌های CL=F استفاده کرده‌ایم. این داده‌ها از تاریخ 1 ژانویه 2010 تا 20 دسامبر 2024 ثبت شده‌اند و در دیتافریم نامپای، شامل ستون‌های زیر می‌باشد:

	Date	Adj Close	Close	High	Low	Open	Volume
0	2010-01-04	81.510002	81.510002	81.680000	79.629997	79.629997	263542
1	2010-01-05	81.769997	81.769997	82.000000	80.949997	81.629997	258887
2	2010-01-06	83.180000	83.180000	83.519997	80.849998	81.430000	370059
3	2010-01-07	82.660004	82.660004	83.360001	82.260002	83.199997	246632
4	2010-01-08	82.750000	82.750000	83.470001	81.800003	82.650002	310377
5	2010-01-11	82.519997	82.519997	83.949997	81.959999	82.879997	296304
6	2010-01-12	80.790001	80.790001	82.339996	79.910004	82.070000	333866
7	2010-01-13	79.650002	79.650002	80.669998	78.370003	80.059998	401627
8	2010-01-14	79.389999	79.389999	80.360001	78.919998	79.629997	275404
9	2010-01-15	78.000000	78.000000	79.309998	77.699997	79.199997	200555

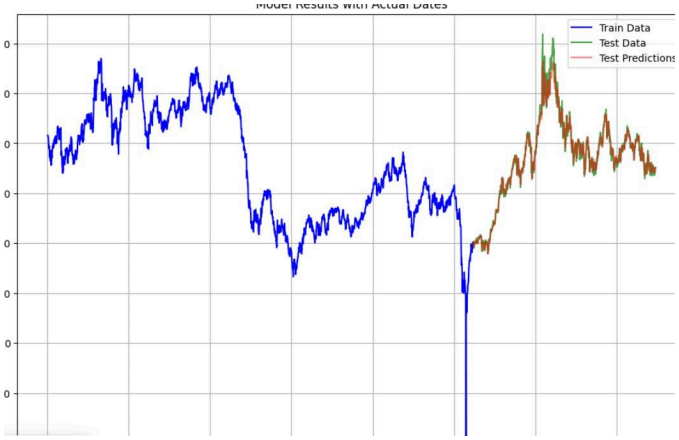
شکل 20. چند نمونه داده از دیتا فریم مسئله

این مجموعه داده شامل اطلاعاتی تاریخی قیمت نفت خام است که در بورس کالای شهر New York[22] معامله می‌شود. این ستون‌ها شامل قیمت باز، بالا، پایین، بسته‌شدن، حجم معاملات، و قیمت بسته‌شدن تعدیل شده می‌باشد.

2-2-2. بررسی مجموعه داده

در برخی از تاریخ‌های این مجموعه داده اطلاعاتی در سطر متناظرشان وجود ندارد. همینطور در بررسی‌ها و اجرای اولیه مدل متوجه شدیم یکی از این سطرها به اشتباه دارای مقداری منفی به ازای Adj Close می‌باشد:

[22] NYMEX



شکل 21. مشاهده یک نمونه داده با مقدار غلط و نتیجه آن بر روی مدل

حال برای شبیه‌سازی دنیای واقعی، 10 درصد از مقادیر مجموعه داده به طور تصادفی روی مقدار تهی[23] تنظیم می‌کنیم. در نهایت همه‌ی داده‌های تنظیم‌شده به تهی با روش‌های معرفی شده قابل جایگزینی می‌باشند.

3-2-2. روش‌های جایگزینی داده‌های Null در مجموعه داده

برای جایگذینی مقادیر نال از روش های مختلفی به کمک متد interpolate کتابخانه پانداس میتوان استفاده کرد که برخی از آنها به شرح زیر اند :

1- linear: در این روش مقادیر گم شده بر اساس خطی مستقیم میان داده های مرزی موجود پر میشوند.

2-time: این روش برای داده های سری زمانی و به کمک زمان به تخمین مقادیر گم شده میپردازد.

3-ffill: مقدار بعدی را جایگذین میکند.

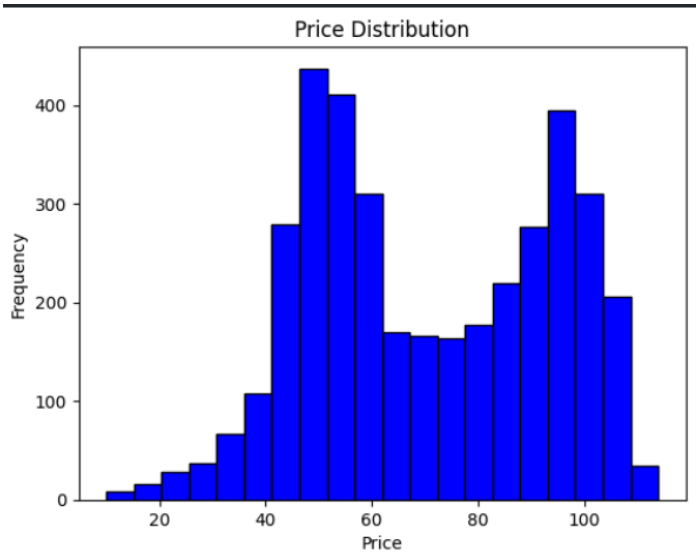
4-bfill: مقدار قبلی را جایگذین میکند.

به دلیل ذات زمانی داده ما میتوان از متد تایم استفاده کرد و همچنین به دلیل بازه های زمانی ثابت میان دیتا پوینت های میس شده عملکرد متد لینیر نیز شرایط دنیای واقعی را به صورت بهتری مدل میکنند.

[23] Null

2-2-4. تقسیم بندی مجموعه داده

دیتای زمانی به دو دسته train و test به نسبت‌های 7 به 3 تقسیم می‌کنیم. از آنجایی که داده ما سری زمانی می‌باشد، 70 درصد اول داده مجموعه train و 30 درصد نهایی مجموعه تست تلقی می‌شود. در این بخش به بررسی توزیع قیمت می‌پردازیم:



شکل 22. توزیع قیمت‌ها و میزان تکرر آن‌ها

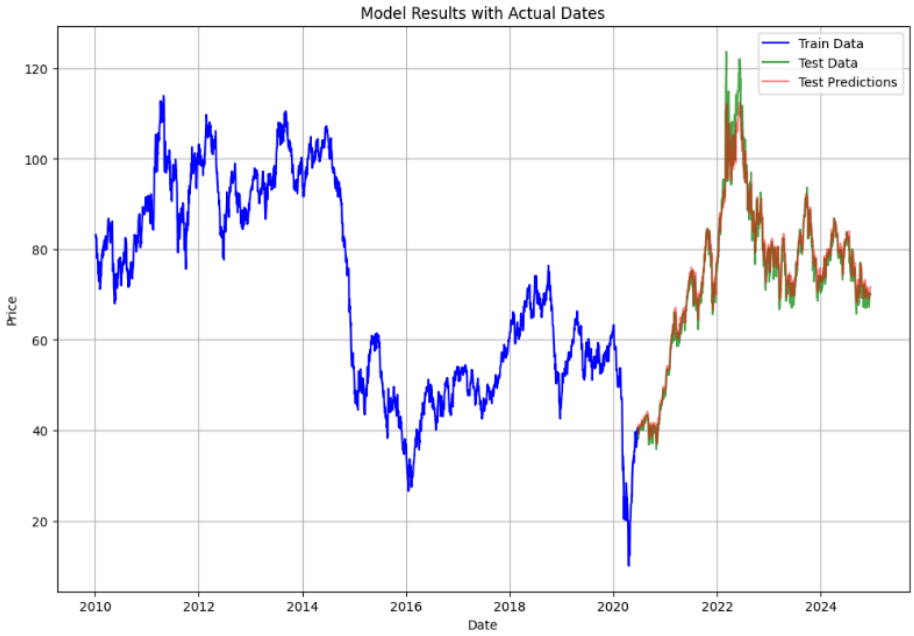
2-3. پیاده‌سازی مدل‌ها

در این بخش به بررسی هایپر پارامترهای مدل‌های استفاده شده می‌پردازیم:

جدول 3. بررسی هایپر پارامترهای مدل‌ها

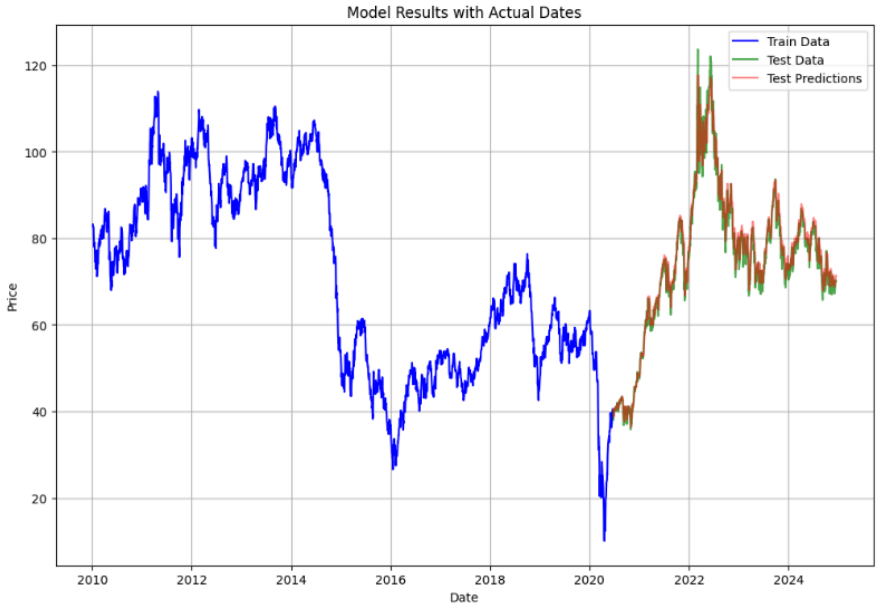
	Units	Batch Size	Loss Function	lr	Drop Out Rate
LSTM	512	100	MSE	0.001	0.2
Bi-LSTM	1024	100	MSE	0.001	0.2
GRU	512	100	MSE	0.001	0.2

2-3-1. مدل LSTM



شکل 23. نتیجه مدل LSTM بر روی مجموعه داده تست

2-3-2. مدل Bi-LSTM



شکل 24. نتیجه مدل Bi-LSTM بر روی مجموعه داده تست

2-3-3. مدل GRU

شکل 25. نتیجه مدل GRU بر روی مجموعه داده تست

2-3-4. متریک‌های ارزیابی

در این مسئله از متریک‌های زیر برای ارزیابی مدل‌ها استفاده شده است:

RMSE[24]: ریشه میانگین مربعات خطا (RMSE) یکی دیگر از معیارهای محبوب برای ارزیابی دقت مدل های رگرسیون است. این جذر میانگین مجذور اختلاف بین مقادیر پیش بینی شده و واقعی را اندازه می گیرد. RMSE به ویژه در سناریوهایی که خطاهای بزرگتر باید به شدت جریمه شوند مفید است. این متریک به مقیاس داده حساس است. فرمول آن به صورت زیر است:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

MAPE[25]: این متریک میانگین بزرگی خطای بین مقادیر پیش‌بینی‌شده و واقعی را که به صورت درصدی از مقادیر واقعی بیان می‌شود، اندازه‌گیری می‌کند. این متریک به مقیاس داده

[24] Root Mean Squared Error

[25] Mean Absolute Percentage Error

حساس نیست، چرا که میزان ارور را به صورت درصدی از مقادیر واقعی بیان می‌دارد. فرمول آن به صورت زیر است:

$$MAPE = \frac{100}{n}\sum_{i=1}^{n} |\frac{y_i - \hat{y}_i}{y_i}|$$

MAE[26]: میانگین خطای مطلق (MAE) یک معیار پرکاربرد برای ارزیابی دقت مدل های رگرسیون است. میانگین بزرگی خطاها را بین مقادیر پیش‌بینی‌شده و واقعی، بدون در نظر گرفتن جهت آنها اندازه‌گیری می‌کند. MAE به ویژه در سناریوهایی که هدف به حداقل رساندن تفاوت های مطلق بین پیش بینی ها و مشاهدات واقعی است مفید است. این متریک به مقیاس داده حساس است. فرمول آن به صورت زیر است:

$$MAE = 100 \times \frac{1}{n}\sum_{i=1}^{n} |\frac{y_i - \hat{y}_i}{y_i}|$$

R2 Score[27]: که به عنوان ضریب تعیین نیز شناخته می شود، یک معیار آماری است که نسبت واریانس متغیر وابسته را که از روی متغیرهای مستقل قابل پیش بینی است، نشان می دهد. این متریک به مقیاس داده حساس نیست. نسبت واریانس متغیر وابسته را که از روی متغیرهای مستقل قابل پیش بینی است اندازه گیری می کند. فرمول آن به صورت زیر است:

$$RSquared = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - y)^2}$$

5-3-2. مقایسه سه مدل

همانطور که مشخص است، هر سه مدل تا حد خوبی قیمت‌ها را در زمان‌های آینده پیش‌بینی کرده‌اند. همینطور به بررسی متریک‌های ارزیابی را بر حسب درصد روی سه مدل می‌پردازیم:

[26] Mean Absolute Error

[27] R-Squared

جدول 2. نتیجه متریک‌های ارزیابی روی مجموعه داده آموزش

Train	LSTM	Bi-LSTM	GRU
MAE	1.34	1.07	1.14
RMSE	1.82	1.47	1.59
R2	99.36	99.58	99.51
MAPE	43.84	43.82	43.43

جدول 3. نتیجه متریک‌های ارزیابی روی مجموعه داده تست

Test	LSTM	Bi-LSTM	GRU
MAE	1.77	1.56	1.60
RMSE	2.54	2.13	2.31
R2	97.77	98.43	98.15
MAPE	27.38	28.10	27.29

همانطور که مشخص است، نتایج روی داده آموزش هر سه مدل بسیار نزدیک به هم می‌باشند. مدلی بهتر از بقیه عمل کرده که مقادیر خطا مانند MAE و RMSE و MAPE کوچک‌تر و مقادیر اسکور R2 بالاتری داشته باشد. با این حال نتایج روی داده تست در مدل Bi-LSTM بهتر از دو مدل دیگر نتیجه داده است. علت این عملکرد هم وابسته به ساختار این مدل است. مدل Bi-LSTM از دو LSTM تشکیل شده است. یکی از این مدل‌ها برای پیش‌بردن داده‌ها در جهت رفت و دیگری در جهت برگشت عمل می‌کند. در نتیجه‌ی این معماری مدل اطلاعات وابستگی‌های طولانی مدت را هم از گذشته به آینده و هم از آینده به گذشته در اختیار دارد. این ویژگی در مسائل با داده‌ی سری زمانی با روندهای نسبتا پیچیده عملکرد خوبی نشان می‌دهد. در مسئله ما که تخمین قیمت

در بازه زمانی از تاریخ است، و به علت ماهیت داده که نوسانات با الگوی نسبتا پیچیده داشته، به خوبی عمل کرده است.

مدل GRU در رتبه‌ی دوم قرار دارد. با این حال مدل LSTM که نسخه‌ی ساده‌تر جفت مدل‌های قبلی‌ست، نتیجه خیلی بدی ندارد. در بخش بعد به طور دقیق‌تر معماری این سه مدل را بررسی می‌کنیم.

6-3-2. توضیح سه مدل

- **مدل LSTM :**

حافظه کوتاه مدت بلند مدت (LSTM) نوعی شبکه عصبی بازگشتی (RNN) است که برای رسیدگی به مشکل گرادیان ناپدید کننده ای که RNN های سنتی با آن مواجه هستند طراحی شده است. LSTM ها به ویژه برای کارهایی که شامل داده های متوالی هستند، مانند پیش بینی سری های زمانی، پردازش زبان طبیعی و تشخیص گفتار موثر هستند.

در این بخش به توضیح ویژگی‌های کلیدی مدل می‌پردازیم. LSTMها دارای سلول‌های حافظه هستند که می‌توانند اطلاعات را در دوره‌های طولانی ذخیره کنند و به شبکه اجازه می‌دهند اطلاعات مهم را از قبل در دنباله به خاطر بسپارند. LSTM ها از سه نوع گیت- ورودی، فراموشی و خروجی- برای کنترل جریان اطلاعات استفاده می کنند. این گیت ها به مدل کمک می کنند تا تصمیم بگیرد کدام اطلاعات را حفظ کند، کدام را دور بیندازد و کدام را خروجی بگیرد. LSTMها می‌توانند وابستگی‌های طولانی‌مدت را در داده‌های متوالی جمع‌آوری کنند، و آنها را برای کارهایی مناسب می‌سازد که زمینه‌های قبلی در توالی بسیار مهم است.

- **مدل Bi-LSTM :**

حافظه کوتاه مدت دو جهته (Bi-LSTM) قابلیت های LSTM های استاندارد را با پردازش داده ها در جهت جلو و عقب گسترش می دهد. این به مدل اجازه می دهد تا زمینه را از هر دو حالت گذشته و آینده بگیرد و توانایی آن را برای درک دنباله افزایش دهد.

در این بخش به توضیح ویژگی‌های کلیدی مدل می‌پردازیم. Bi-LSTM ها از دو لایه LSTM تشکیل شده اند- یکی از ابتدا تا انتها (به جلو) و دیگری از انتهای به شروع (به عقب) پردازش می کند. این رویکرد دوسویه درک جامع تری از توالی ارائه می دهد. با در نظر گرفتن زمینه گذشته و آینده، Bi-LSTM ها می‌توانند وابستگی‌ها و روابط ظریف‌تری را در داده‌ها ثبت کنند. Bi-LSTM ها اغلب در وظایفی که زمینه از هر دو جهت مفید است، مانند طبقه بندی متن و شناسایی موجودیت نامگذاری شده، بهتر از LSTM های استاندارد عمل می کنند.

- **مدل GRU :**

واحد بازگشتی در دار (GRU) نوعی از RNN است که معماری LSTM ها را ساده می کند و در عین حال توانایی آنها را برای گرفتن وابستگی های طولانی مدت حفظ می کند. GRU ها برای رسیدگی به مشکل گرادیان ناپدید طراحی شده اند و اغلب به عنوان جایگزینی برای LSTM ها استفاده می شوند.

در این بخش به توضیح ویژگی‌های کلیدی مدل می‌پردازیم. GRU ها در مقایسه با LSTM ها ساختار ساده تری دارند و تنها دو گیت دارند- گیت های بازنشانی و به روز رسانی. این امر پیچیدگی محاسباتی و زمان آموزش را کاهش می دهد. GRU ها حالت سلولی و حالت پنهان را در یک حالت واحد ترکیب می‌کنند و باعث می‌شوند حافظه کارآمدتری داشته باشند. علیرغم معماری ساده تر، GRU ها اغلب به عملکرد قابل مقایسه با LSTM ها در وظایف مختلف داده های متوالی دست می یابند.

2-4. ARIMA

2-4-1. تفاوت مدل‌های ARIMA و SARIMA

این دو مدل از مدل‌های کلاسیک در تحلیل داده‌های سری زمانی هستند که در محاسبات ریاضی خود از سه پارامتر استفاده می‌کنند.

این دو مدل از جهت‌هایی با هم تفاوت دارند، مانند اینکه مدل ARIMA برای سری‌های زمانی غیر ایستا استفاده می‌شود. این سری‌های زمانی داده‌هایی هستند که روند و الگوی فصلی ندارند. از آنجایی که ورودی مدل برای بخش‌های AR و MA داده‌های غیر ایستاست، از بخش Integrated مدل برای تبدیل روندهای زمانی ورودی را به حالت ایستا استفاده می‌کند.

در حالی‌که مدل SARIMA برای برای داده‌های سری زمانی Seasonal استفاده می‌شود. این مدل علاوه بر مولفه‌های مدل ARIMA، به تعداد 4 مولفه فصلی دارد که به مدل قابلیت شبیه‌سازی و پیش‌بینی روی داده‌های دارای نوسانات یا الگوهای فصلی را می‌دهد:

Seasonal AR - Seasonal I - Seasonal MA - Seasonal Period

بنابراین مدل SARIMA پیچیده‌تر از ARIMA بوده و هزینه‌ی زمانی بیشتری دارد. در این مسئله ما به علت ماهیت مجموعه داده‌مان از مدل ساده‌تر ARIMA استفاده کرده‌ایم.

2-4-2. مزایا و محدودیت‌های مدل ARIMA

از مزایای مدل ARIMA می‌توان به ساختار نسبتا ساده و قابل درک مدل اشاره کرد که به طور گسترده در پیشبینی سری‌های زمانی استفاده می‌شود. این مدل برای پیش‌بینی کوتاه‌مدت حتی روی داده‌های نویزدار کاربرد خوبی دارد. مدل با سادگی خود نیازی به متغیرهای برون زا یا Exogenous Variables ندارد و صرفا روی مقادیر تاریخی سری‌های زمانی تکیه می‌کند.

از معایب مدل می‌توان به این اشاره کرد که روی داده‌های غیر ثابت[28] یا ضعیف ایده‌آل نبوده و عملکرد خوبی نشان نمی‌دهد. همینطور مدل می‌تواند دچار برازش بیش از حد[29] شود، مخصوصا در حالت‌های که پیش‌بینی طولانی مدت نیاز است. این مدل نمی‌تواند داده‌های فصلی یا Seasonal را مدیریت کند. بنابراین روی داده‌های دارای الگوی فصلی قوی انتخاب مناسبی نمی‌باشد.

2-4-3. تعریف ریاضی مدل

این مدل از 3 پارامتر اصلی p و q و d تشکیل شده است[30]. مدل ریاضی ARIMA به کمک این پارامترهای ورودی به پیش‌بینی سری‌های زمانی ایستا می‌پردازد. در بخش زیر به تشریح پارامترهای مدل می‌پردازیم:

- پارامتر p یا همان order of the AutoRegressive part[31]، که نشان‌دهنده‌ی
- پارامتر q یا همان order of the moving average[32]،
- پارامتر q یا همان degree of difference to achieve stationarity، می‌باشند.

از لحاظ ریاضی مدل ARIMA به صورت زیر نمایش داده می‌شود:

---

[28] Non-Stationary Data

[29] Overfitting

[30] ARIMA(p, d, q)

[31] AR

[32] MA

$$Y_t = c + \varphi_1 Y_{t-1} + \ldots + \varphi_p Y_{t-p} - \theta_1 \epsilon_{t-1} - \ldots - \theta_q \epsilon_{t-q} + \epsilon_t$$

φ_i: $Autoregressive\ coefficients$

θ_i: $Moving\ average\ coefficients$

d: $Differencing\ degree\ to\ achieve\ stationarity$

ϵ_t: $Error\ term\ (white\ noise)$

که در آن c میانگین سری زمانی می‌باشد.

2-4-4. پارامترهای بهینه مدل

برای به دست آوردن پارامترهای بهینه مدل ARIMA از کتابخانه‌ای به اسم pmdarima استفاده کرده‌ایم. این پارامترهای بهینه بر حسب داده‌ی ورودی و توزیع آن مشخص می‌شود.

- Auto-Arima Algorithm:

این کتابخانه با این الگوریتم به کاربرانش این امکان را می‌دهد که به طور خودکار بهترین مدل ARIMA را برای داده‌های سری زمانی خود انتخاب کنند.

- Statistical Tests:

این کتابخانه شامل انواع آزمون‌های آماری برای ثابت بودن و فصلی بودن است که به کاربرانش این امکان را می‌دهد تا ویژگی‌های داده‌های سری زمانی خود را درک کنند.

- Time Series Utilities:

ابزارهایی مانند تفاضل[33]، تفاضل معکوس[34]، تجزیه فصلی[35] برای پیش پردازش و تجزیه و تحلیل داده‌های سری زمانی در دسترس هستند.

2-4-5. الگوریتم Auto Arima

حال به تشریح الگوریتم Auto Arima می‌پردازیم. این الگوریتم بر اساس معیار AIC[36] بهترین پارامترها را برای مدل با توجه به مجموعه داده ورودی انتخاب می‌کند که در عین حال مدل دچار Overfitting روی این مجموعه داده نشود. در واقع AIC یک معیار آماری بوده که برای ارزیابی

[33] Differencing

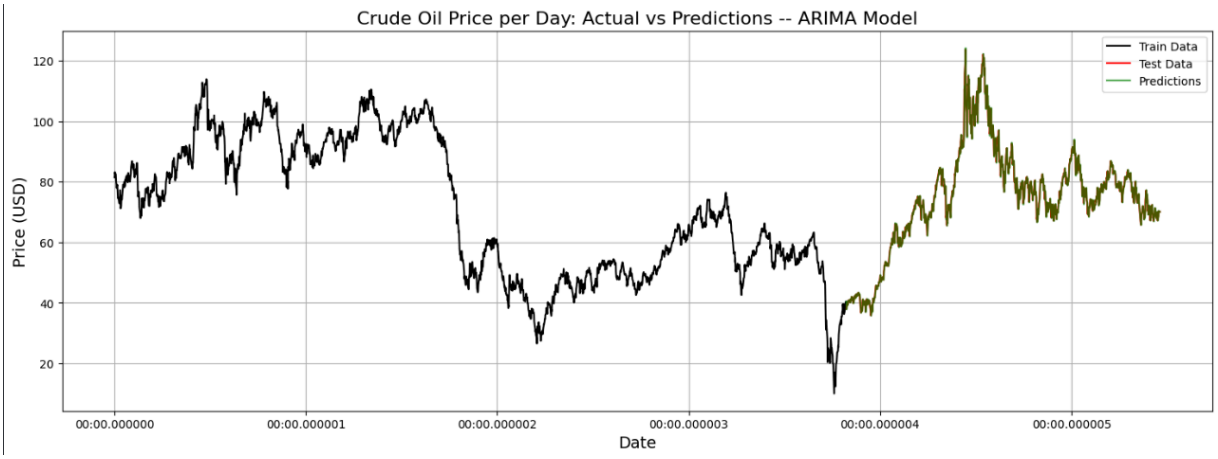
[34] Inverse Differencing

[35] Seasonal Decomposition

[36] Akaike Information Criterion

کیفیت مدل‌های آماری مانند Regression Models و مدل‌های مربوط به سری‌های زمانی استفاده شده کاربرد دارد.

. عکس کد



بهترین پارامترها برای مدل فوق به صورت زیر تعیین شده است.

$p =, q =, d =$

حالال

6-4-2. نتیجه اجرای مدل

جدول 4. نتیجه متریک‌های ارزیابی روی مجموعه داده تست

Test	LSTM	Bi-LSTM	GRU	ARIMA
MAE	1.77	1.56	1.60	0.86
RMSE	2.54	2.13	2.31	1.37
R2	97.77	98.43	98.15	99.34
MAPE	27.38	28.10	27.29	1.13

همانطور که مشخص است، مدل ARIMA روی داده‌ی تست، بسیار بهتر از سه مدل قبلی عمل کرده است. این مدل روی متریک‌های خطا مانند RMSE ،MAE، و MAPE میزان خطای کمتری ثبت کرده‌است. همینطور متریک R2 Score آن مقداری بیشتری از سه مدل قبل دارد. علت این

اتفاق هم در ساختار مدل می‌باشد. این مدل از قابلیت rolling forecast بهره می‌برد که در نتیجه این اتفاق مدل مدام عملکرد خود را نسبت به داده‌ی جدیدی که دیده بهبود می‌دهد و در نتیجه یاد می‌گیرد که پیش‌بینی دقیق‌تری انجام دهد.

ابتدا دو جدول را نمایش می‌دهیم:

جدول 5. نتیجه متریک‌های ارزیابی روی مجموعه داده تست

Test	MAE	RMSE	R2	MAPE (%)
LSTM	1.77	2.54	97.77	27.38
Bi-LSTM	1.56	2.13	98.43	28.10
GRU	1.60	2.31	98.15	27.29
ARIMA	0.86	1.37	99.34	1.13

جدول 6. جدول شماره 6 از مقاله

Table 6. MAE, RMSE, R-Squared, and MAPE of Crude Palm Oil Price in Medan

Method	MAE	RMSE	R-squared	MAPE (%)
• **LSTM**	**667.6389**	**896.7875**	**0.9425**	**3.9412**
• **LSTM** [34]	**429.8600**	**568.2900**	**0.9100**	**2.7800**
• **GRU**	**531.0581**	**718.0000**	**0.9631**	**3.0547**
• Bi-LSTM	878.4119	1,108.9177	0.9120	5.2493
• ARIMA (1,1,5) [14]	4,700.3681	5,817.1785	-0.011	35.9599
• ARIMA (2,2,2) [14]	4,898.0502	5922.1584	-0.0476	36.6571
• Simple RNN [11]	615.5000	808.4800	0.8100	4.1200
• SVR [13]	4,848.7671	5,807.5774	-0.0109	47.3958

جدول 7. جدول شماره 7 از مقاله

Table 7. MAE, RMSE, R-Squared, and MAPE of Crude Palm Oil Price in Rotterdam (SPOT)

Method	MAE	RMSE	R-squared	MAPE (%)
•LSTM	52.1434	74.6622	0.9431	4.2732
•**LSTM** [34]	**436.2500**	**578.0000**	**0.9000**	**2.8500**
•GRU	37.5861	57.7891	0.9659	3.0493
•Bi-LSTM	55.4324	77.5116	0.9387	4.5689
•**ARIMA (1,1,5)** [14]	**14.0376**	**43.5947**	**0.9846**	**2.4500**
•**ARIMA (2,2,2)** [14]	**14.4271**	**45.4282**	**0.9833**	**2.5126**
•Simple RNN [11]	38.5600	50.8500	0.3900	3.8100
•SVR [13]	279.4392	343.3406	0.0460	34.9876

همانطور که در صورت تمرین ذکر شده، مجموعه داده مورد بررسی در این دو مسئله متفاوت بوده و دلیل تفاوت مقیاس خطا و اسکور بین دو جدول به همین علت می‌باشد. بنابراین به سراغ مقایسه دو متریک R2 و MAPE می‌رویم که به مقیاس داده‌ها حساس نیستند.