Cofee Bazar Comment Classification

Project-Link

Alireza Toomari

01

Look data

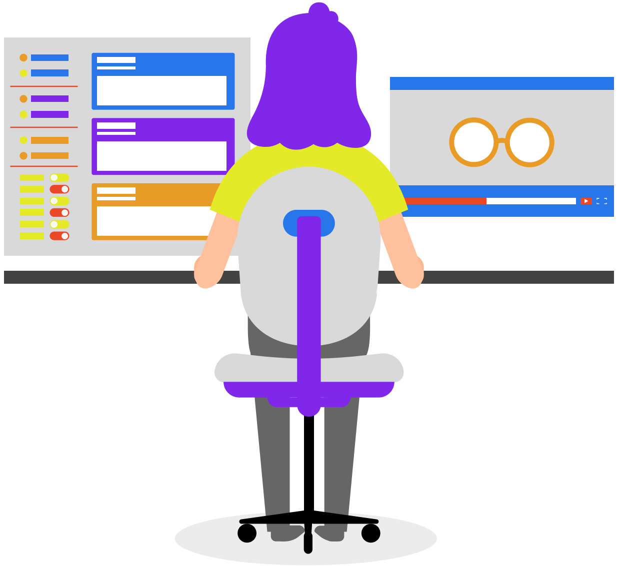
02

Analyze and Prepare Data

Imbalance data

03

Choose model archtecture

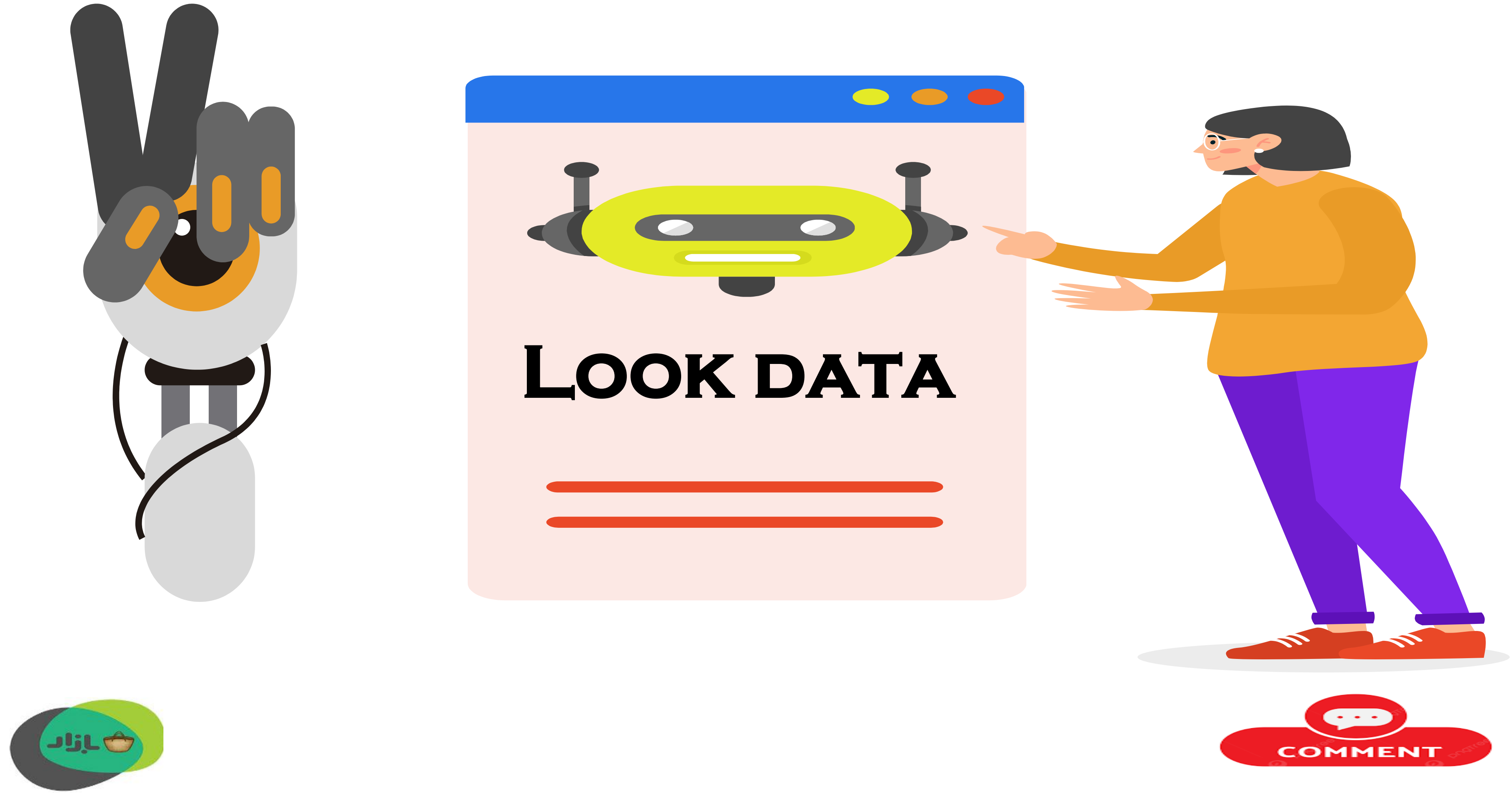


04

Models Report

05

Conclusion


LOOK DATA
بازار
COMMENT

Dataset contains comments of **CofeeBazar** that each comment has True False label for classes .

INFO :

- Number of Comments with NAN labels: 162276
- Number of Comments without NAN labels:111011
- Number of Classes : 9

Classes →

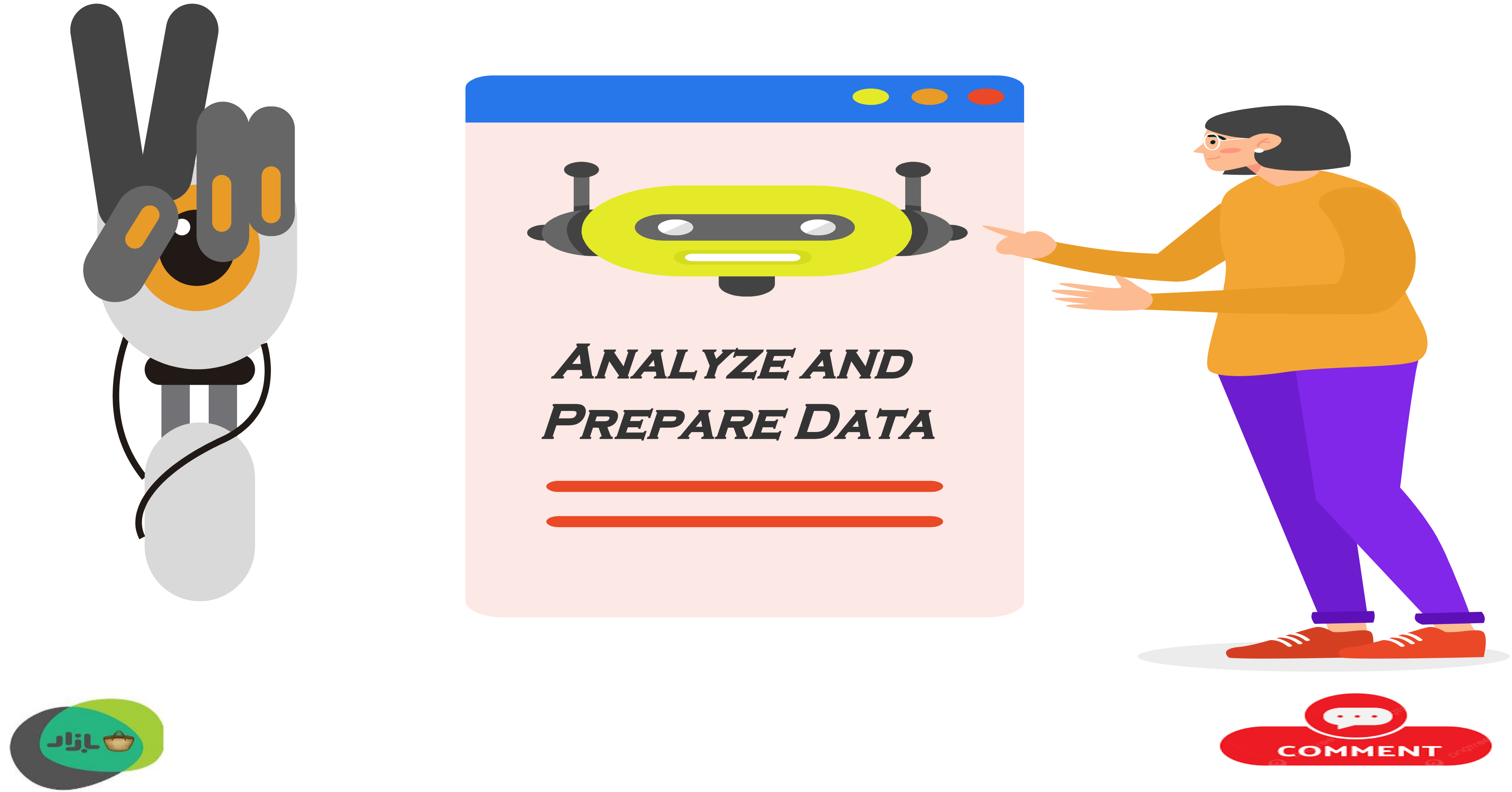
مقایسه با رقبا	پیشنهاد نرم افزاری	درخواست بروز رسانی	اشکال فنی	اعتراض به خدمات	درخواست راهنمایی	اعتراض به قیمت	مشکل در پرداخت	حجم زیاد نرم افزار

70% of Comments have labels with zero values so we use just 30/70 number of this comments for faster training .

Class Names	مقایسه با رقبا	پیشنهاد نرم افزاری	درخواست بروز رسانی	اشکال فنی	اعتراض به خدمات	درخواست راهنمایی	اعتراض به قیمت	مشکل در پرداخت	حجم زیاد نرم افزار
Count	2847	4751	1983	8379	1208	3547	5796	240	522

Data is imbalance , In the next part we’ll try to make it Balance .

Notice : After reducing ZERO labels now we have 39984 comments .


ANALYZE AND PREPARE DATA
COMMENT

As we saw, our data was imbalanced, so we used weighting classes to solve this challenge
To do this , we compute classes weight and use on of these 2 methods :

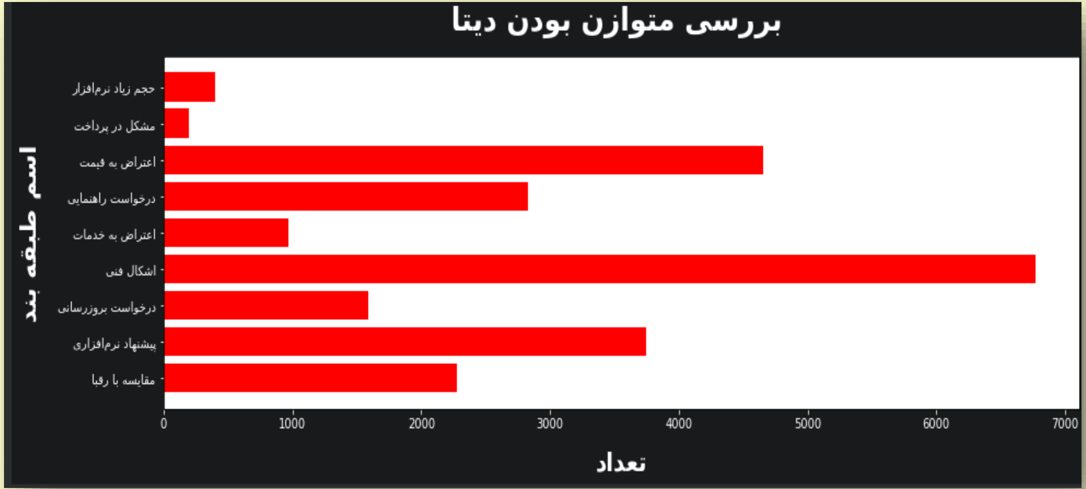
- Use Class Weights in training phase
- Use Custom Loss

```python
def compute_class_weights(y):
    class_weights =  tf.reduce_sum(tf.ones_like(y), axis=0) / tf.reduce_sum(y, axis=0)
    class_weights = class_weights / (K.max(class_weights) + K.epsilon())  # add epsilon

    return tf.cast(class_weights, tf.float32)


weights = compute_class_weights(y_train)

def weighted_binary_crossentropy(class_weights):
    def loss(y_true, y_pred):
        # Apply class weights to the binary cross-entropy loss for each label
        losses = tf.nn.sigmoid_cross_entropy_with_logits(labels=y_true, logits=y_pred)
        weighted_losses = tf.reduce_mean(class_weights * losses, axis=0)

        # Return the mean of the weighted losses
        return tf.reduce_mean(weighted_losses)
    return loss
```

COMMENT

Imbalanced Classes

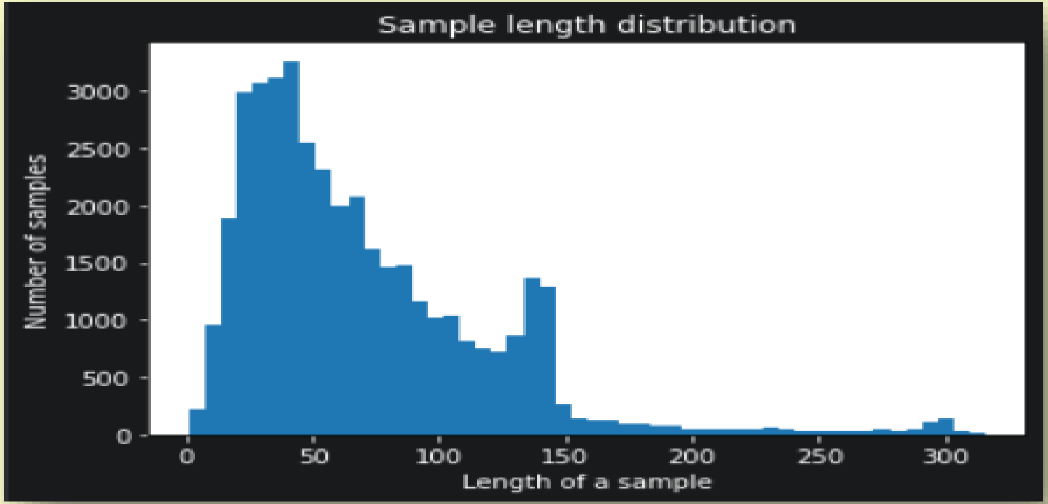
Well, now it's time to analyze the data
To do this ,first we visualize the class weights
Then showing the mean average of comments length
Ok let's start ,



	cat	count	class_weight
0	مقایسه با رقبا	2847.0	0.003161
1	پیشنهاد نرم‌افزاری	4751.0	0.001894
2	درخواست بروزرسانی	1983.0	0.004539
3	اشکال فنی	8379.0	0.001074
4	اعتراض به خدمات	1208.0	0.007450
5	درخواست راهنمایی	3547.0	0.002537
6	اعتراض به قیمت	5796.0	0.001553
7	مشکل در پرداخت	240.0	0.037500
8	حجم زیاد نرم‌افزار	522.0	0.017241

Sample Length Distribution

As you see most of the comments are lower than 150



Preparing Data

First we need to cleaning comments and normalize data
that I have used Persian text normalization methods

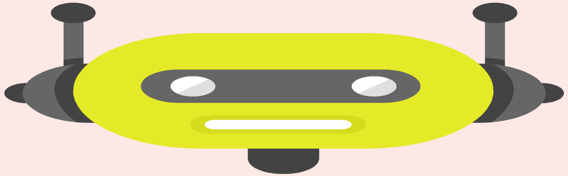
To feed the model with data ,we need to tokenizing it to sequences with numbers And
if data length is shorter than max length padding it with zero and else truncating it .
In this project because of that I’ve used transformer model , the max length is
the default of model **136**

Example : خیلی خوبه ولی بیش از حد سخته

54	75	25	46	78	32	23	0	......	0

136

COMMENT



Choose model archtecture

Models

The models are based of Bert model that is trained on Persian texts ,comments
This Bert model Transformer is on Hugging Face

Bert Model Name : **'HooshvareLab/bert-fa-zwnj-base'**

With consider this problem I've trained 4 models
- Bert Model (using pooled output of bert model)
- SepCNN Model (using embedding layer of bert with Seprable Convolution layer)
- RnnCNN Model (using pooled output of bert layer with LSTM and CNN)
- MLP Model (Using Embedding Layer of bert with Dense layer)

BERT Model

```python
class BertModel(tf.keras.Model):

    def __init__(self, bert,config ,dense_layers_num = 0):
        super(BertModel, self).__init__()

        self.bert_layer = bert
        self.dropout = tf.keras.layers.Dropout(config.hidden_dropout_prob)
        self.classifier = tf.keras.layers.Dense( config.num_labels, activation='sigmoid')
        self.dropout_layers = []
        self.dense_layers = []

        for i in range(dense_layers_num):
          self.dropout_layers.append(tf.keras.layers.Dropout(rate = config.hidden_dropout_prob))
          self.dense_layers.append(tf.keras.layers.Dense(units=100, activation="relu" ,kernel_regularizer=regularizers.l2(0.01)))
    def call(self, inputs):
        input_ids, attention_mask = inputs['input_ids'] ,inputs['attention_mask']

        x = self.bert_layer(input_ids, attention_mask=attention_mask)
        x = self.dropout(x['pooler_output'])
        for dropout_layer, dense_layer in zip(self.dropout_layers, self.dense_layers):
            x = dense_layer(x)
            x = dropout_layer(x)
        logits = self.classifier(x)
        return logits


    def freeze_bert_layers(self):
        self.bert_layer.trainable = False

    def unfreeze_bert_layers(self):
        self.bert_layer.trainable = True
```

MLP Model

```
class MLPModel(tf.keras.Model):
    def __init__(self ,bert_layer , layers, units, dropout_rate, num_classes, input_shape):
        super(MLPModel, self).__init__()

        self.bert_layer = bert_layer
        self.dropout_layers = []
        self.dense_layers = []
        for i in range(layers):
            self.dropout_layers.append(tf.keras.layers.Dropout(rate=dropout_rate))
            self.dense_layers.append(tf.keras.layers.Dense(units=units, activation="relu" ,kernel_regularizer=regularizers.l2(0.01)))
        self.output_layer = tf.keras.layers.Dense(units=num_classes, activation="sigmoid")
        self.pool_layer = tf.keras.layers.GlobalMaxPool1D()

    def call(self, inputs, **kwargs):
        input_ids, attention_mask = inputs['input_ids'] ,inputs['attention_mask']
        bert_output = self.bert_layer(input_ids, attention_mask=attention_mask)[0]
        out = self.pool_layer(bert_output)
        for dropout_layer, dense_layer in zip(self.dropout_layers, self.dense_layers):
            out = dense_layer(out)
            out = dropout_layer(out)
        out = self.output_layer(out)
        return out

    def freeze_bert_layers(self):
        self.bert_layer.trainable = False

    def unfreeze_bert_layers(self):
        self.bert_layer.trainable = True
```

SepCNN Model

```python
class SepCNNModel(tf.keras.Model):

    def __init__(self, bert ,config,blocks, filters, kernel_size,
                 pool_size, num_classes  ,trainable = False):

        super(SepCNNModel, self).__init__()

        self.blocks = blocks
        self.filters = filters
        self.kernel_size = kernel_size
        self.pool_size = pool_size
        self.num_classes = num_classes
        self.trainable = trainable
        self.bert_layer = bert
        self.block_layers = []
        self.sep_cnn_layers = []

        for i in range(self.blocks):
            self.block_layers.append(tf.keras.layers.Dropout(config.hidden_dropout_prob))
            self.block_layers.append(tf.keras.layers.SeparableConv1D(filters=self.filters,
                                                                     kernel_size=self.kernel_size,
                                                                     activation='relu',
                                                                     bias_initializer='random_uniform',
                                                                     depthwise_initializer='random_uniform',
                                                                     padding='same'))
            self.block_layers.append(tf.keras.layers.SeparableConv1D(filters=self.filters,
                                                                     kernel_size=self.kernel_size,
                                                                     activation='relu',
                                                                     bias_initializer='random_uniform',
                                                                     depthwise_initializer='random_uniform',
                                                                     padding='same'))
            self.block_layers.append(tf.keras.layers.MaxPooling1D(pool_size=self.pool_size))

        self.sep_cnn_layers.append(tf.keras.layers.SeparableConv1D(filters=self.filters * 2,
                                                                   kernel_size=self.kernel_size,
                                                                   activation='relu',
                                                                   bias_initializer='random_uniform',
                                                                   depthwise_initializer='random_uniform',
                                                                   padding='same'))
        self.sep_cnn_layers.append(tf.keras.layers.SeparableConv1D(filters=self.filters * 2,
                                                                   kernel_size=self.kernel_size,
                                                                   activation='relu',
                                                                   bias_initializer='random_uniform',
                                                                   depthwise_initializer='random_uniform',
                                                                   padding='same'))

        self.global_pooling_layer = tf.keras.layers.GlobalAveragePooling1D()
        self.dropout_layer = tf.keras.layers.Dropout(config.hidden_dropout_prob)
        self.dense_layer = tf.keras.layers.Dense(config.num_labels, activation='sigmoid')

    def call(self, inputs):
        input_ids, attention_mask = inputs['input_ids'] ,inputs['attention_mask']

        # Define the forward pass of the model

        x = self.bert_layer(input_ids, attention_mask=attention_mask)[0]
        for layer in self.block_layers:
            x = layer(x)
        for layer in self.sep_cnn_layers :
            x = layer(x)

        x = self.global_pooling_layer(x)
        x = self.dropout_layer(x)

        x = self.dense_layer(x)

        self.bert_layer.trainable = self.trainable

        return x
```

RNN-CNN Model

```python
class RNN_CNN_Model(tf.keras.Model):
    def __init__(self, bert_layer ,conv_layers_num ,rnn_layers_num ,dense_layers_num ,rnn_units ,dense_units,filters , dropout_rate, num_classes):
        super(RNN_CNN_Model, self).__init__()
        self.bert_layer = bert_layer
        self.reshape_layer = tf.keras.layers.Reshape((1, -1))
        self.num_classes = num_classes
        self.rnn_layers = []
        self.conv_layers = []
        self.dropout_layers = []
        self.dense_layers = []
        self.pool_layer = tf.keras.layers.GlobalMaxPooling1D()
        # Define the layers of the model

        for i in range(rnn_layers_num):
            self.rnn_layers.append( tf.keras.layers.LSTM(rnn_units, return_sequences=True) )

        for i in range(conv_layers_num):
            self.conv_layers.append(tf.keras.layers.Conv1D( filters=filters, kernel_size=3, padding='same', activation='relu'))

        for i in range(dense_layers_num):
            self.dropout_layers.append(tf.keras.layers.Dropout(rate=dropout_rate))
            self.dense_layers.append(tf.keras.layers.Dense(units=dense_units, activation="relu" ,kernel_regularizer=regularizers.l2(0.01)))


        self.output_layer = tf.keras.layers.Dense(self.num_classes, activation='sigmoid')

    def call(self, inputs):
        input_ids, attention_mask = inputs['input_ids'] ,inputs['attention_mask']

        # Define the forward pass of the model

        x = self.bert_layer(input_ids, attention_mask=attention_mask)[1]
        x = self.reshape_layer(x)
        for rnn_layer in self.rnn_layers :
          x = rnn_layer(x)

        for conv_layer in self.conv_layers :
          x = conv_layer(x)

        x = self.pool_layer(x)
        for dropout_layer, dense_layer in zip(self.dropout_layers, self.dense_layers):
            x = dense_layer(x)
            x = dropout_layer(x)


        output = self.output_layer(x)
        return output
```

Select Best Model

I trained the models by **freezing** and **unfreezing**, and according to the results, I used only 3 of them for testing.

1. **BertModel with 1 dense layer (Unfreezed)**

2. **BertModel with 2 dense layer (Unfreezed)**

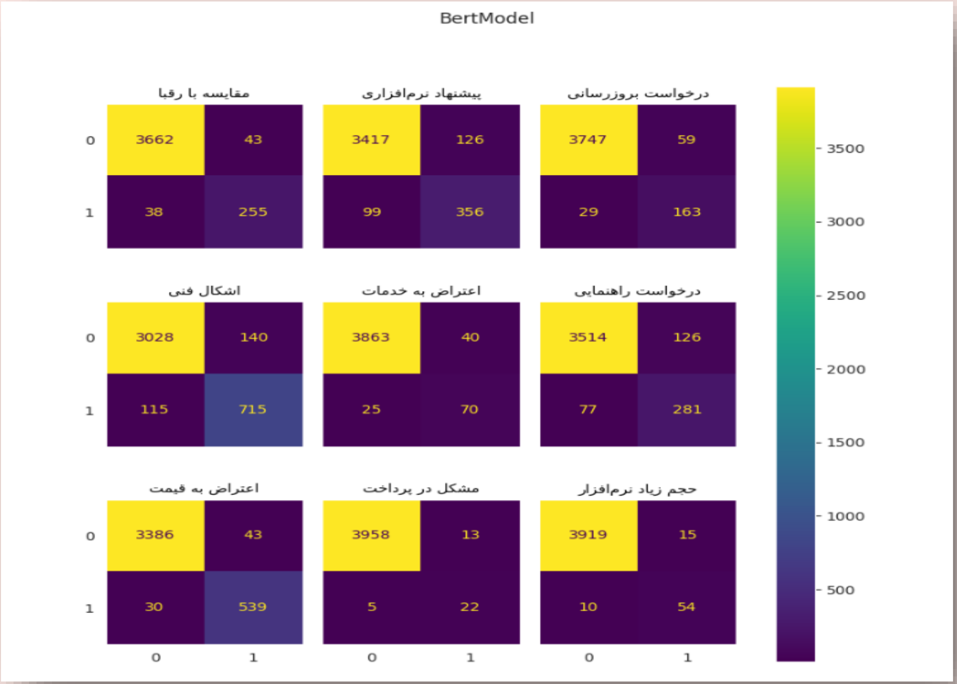
3. **MLP Model (Unfreezed)**

In the next slides we’ll check our results on this models .

Report Models

- BertModel with 1 dense layer (Unfreezed)
- BertModel with 2 dense layer (Unfreezed)
- MLP Model (Unfreezed)

BertModel with 1 dense layer (Unfreezed)



	precision	recall	f1-score	support
مقایسه با رقبا	0.855704698	0.8703071672	0.8629441624	293
پیشنهاد نرم‌افزاری	0.7385892116	0.7824175824	0.7598719317	455
درخواست بروزرسانی	0.7342342342	0.8489583333	0.7874396135	192
اشکال فنی	0.8362573099	0.8614457831	0.8486646884	830
اعتراض به خدمات	0.6363636364	0.7368421053	0.6829268293	95
درخواست راهنمایی	0.6904176904	0.7849162011	0.7346405229	358
اعتراض به قیمت	0.9261168385	0.9472759227	0.9365768897	569
مشکل در پرداخت	0.6285714286	0.8148148148	0.7096774194	27
حجم زیاد نرم‌افزار	0.7826086957	0.84375	0.8120300752	64
micro avg	0.8022875817	0.851543531	0.8261820629	2883
macro avg	0.7587626381	0.8323031011	0.7927524592	2883
weighted avg	0.8059275244	0.851543531	0.8276404048	2883
samples avg	0.5817075204	0.5824162081	0.577980657	2883

COMMENT

BertModel with 2 dense layer (Unfreezed)

	precision	recall	f1-score	support
مقایسه با رقبا	0.8081761006	0.8771331058	0.8412438625	293
پیشنهاد نرم‌افزاری	0.7213483146	0.7054945055	0.7133333333	455
درخواست بروزرسانی	0.6888888889	0.8072916667	0.7434052758	192
اشکال فنی	0.8440366972	0.7759036145	0.8085373509	830
اعتراض به خدمات	0.6304347826	0.6105263158	0.6203208556	95
درخواست راهنمایی	0.6937172775	0.7402234637	0.7162162162	358
اعتراض به قیمت	0.9384057971	0.9103690685	0.9241748439	569
مشکل در پرداخت	0.7916666667	0.7037037037	0.7450980392	27
حجم زیاد نرم‌افزار	0.74	0.578125	0.649122807	64
micro avg	0.797614872	0.7887617066	0.7931635856	2883
macro avg	0.761852725	0.7454189382	0.7512725094	2883
weighted avg	0.8008172717	0.7887617066	0.7935219605	2883
samples avg	0.5537768884	0.5437301984	0.5451809238	2883

MLP Model (Unfreezed)

	precision	recall	f1-score	support
مقایسه با رقبا	0.8220064725	0.866894198	0.8438538206	293
پیشنهاد نرم‌افزاری	0.7330210773	0.6879120879	0.7097505669	455
درخواست بروزرسانی	0.7009345794	0.78125	0.7389162562	192
اشکال فنی	0.8363858364	0.8253012048	0.8308065494	830
اعتراض به خدمات	0.5619834711	0.7157894737	0.6296296296	95
درخواست راهنمایی	0.6585365854	0.7541899441	0.703125	358
اعتراض به قیمت	0.9030100334	0.9490333919	0.9254498715	569
مشکل در پرداخت	0.72	0.6666666667	0.6923076923	27
حجم زیاد نرم‌افزار	0.8032786885	0.765625	0.784	64
micro avg	0.7865281501	0.8140825529	0.8000681779	2883
macro avg	0.7487951938	0.779184663	0.7619821541	2883
weighted avg	0.7897881521	0.8140825529	0.8007665224	2883
samples avg	0.5552359513	0.5577372019	0.551875938	2883

COMMENT

Conclusion

There are other features that you can add to this Project :

You are able to :

- **Using different transformers and Ensembling them .**

- **Using more comments if you have access to a powerful system, as seen in this project, we reduced the data in a specific way.**