Metode

Metode yang kami gunakan pada Aplikasi Chatbot yang sudah kami buat adalah penearapan tokenization agar dapat memudahkan dalam olah kata. Setiap kata yang di tokenisasi juga akan dikumpulkan berdasarkan “tag” (sifatnya).

```
for intent in intents['intents']:
    for pattern in intent['patterns']:

        # mentokenisasi setiap kata
        w = nltk.word_tokenize(pattern)
        words.extend(w)
        # menambahkan documents kedalam corpus
        documents.append((w, intent['tag']))

        # menambahkan ke list classes kita
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Kami juga menerapkan Lemmatize untuk setiap kata dilakukan supaya kata yang sudah berubah karena penempatannya, bisa dikembalikan ke kata dasarnya. Kata tersebut juga diubah menjadi huruf kecil semua (*function lower()*) agar mudah dalam pencocokan kata dan juga dibuang beberapa simbol yang tidak dibutuhkan. Selanjutnya kata tersebut akan diurutkan dan dimasukkan kedalam variabel words. Begitu juga dengan class, kita urutkan dan masukkan ke dalam variabel classes.

```
# lemmaztize and lower each word and remove duplicates
words = [lemmatizer.lemmatize(w.lower())
         for w in words if w not in ignore_words]
words = sorted(list(set(words)))
# sort classes
classes = sorted(list(set(classes)))
# documents = combination between patterns and intents
print(len(documents), "documents")
# classes = intents
print(len(classes), "classes", classes)
# words = all words, vocabulary
print(len(words), "unique lemmatized words", words)
```

Kami menerapkan konsep *bag of word* sebagai media yang bertujuan untuk memudahkan kita dalam membuat "key" terhadap kata yang diucapkan client kepada chatbot.

```
# training set, bag of words untuk setiap kalimat
for doc in documents:
    # menginsialisasi bag of words
    bag = []
    # list dari tokenisasi kata untuk setiap pola
    pattern_words = doc[0]
    # lemmatize setiap kata - buat kata dasar, agar menjadi "key" bagi bentuk kata ubahannya
    pattern_words = [lemmatizer.lemmatize(
        word.lower()) for word in pattern_words]
    # membuat bag of words array dengan 1, jika kecocokan kata ditemukan dalam pola saat ini
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output '0' untuk setiap 'tag' dan '1' untuk 'tag' saat ini (untuk setiap pola)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
```

Setelah itu, kami menerapkan model Sequential untuk metode uji training data nya. . Pengujian ini dimaksudkan untuk menilai keakuratan data terhadap hasil yang diinginkan. Buat 3 Layer model dengan layer pertama berisi 128 neurons, layers ke-2 berisi 64 neuron, dan layer ke-3 berisi output berdasarkan jumlah neuron dan menggunakan fungsi softmax.

```
# buat model - 3 layers. Layer pertama 128 neurons,
# layer kedua 64 neurons
# Layer ke-3 output layer mengandung jumlah neurons
# sama dengan jumlah maksud untuk memprediksi maksud keluaran dengan softmax
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
```

Selanjutnya data train yang sudah di uji akurasinya, akan kita compile menggunakan algoritma SGD. SGD dipilih dikarenakan memiliki fungsi melakukan proses pembelajaran dan akan update setiap ada 1 data baru. Sehingga ini akan membuat model training lebih akurat dan efisien.

```
# Compile model. Penurunan gradien stokastik dengan gradien
# akselerasi Nesterov memberikan hasil yang baik untuk model ini
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd, metrics=['accuracy'])
```