SECTOR CLASSIFICATION FOR CROWD-BASED SOFTWARE REQUIREMENTS

Kushagra Bhatia* and Arpit Sharma
*kushagra1198@gmail.com

TABLE OF CONTENTS

I. Dataset

A. Sample Requirements

Role	**Feature**	**Benefit**	**Application Domain**	**Application domain_other**	**Tags**
worker	my smart home to be able to order delivery food by simple voice command	I can prepare dinner easily after a long day at work	Health		food, delivery, dinner, voice
home occupant	my smart home to turn on certain lights at dusk	I can come home to a well-lit house	Energy		lights, turn on, night
worker	my smart home to sync with my biorhythm app and turn on some music that might suit my mood when I arrive home from work	I can be relaxed	Entertainment		music, biorhythm, mood
parent	my smart home to keep me up to date about my children's activities when I'm out of the home	I can know they're safe and positively occupied	Safety		child, monitor, status
home occupant	my smart home to sync all Christmas-related lights, indoor and outdoor, and turn them on and off at the appropriate time	it's more convenient for me	Other	Convenience	lights, turn on, turn off, indoor, outdoor, Christmas, synchronize

B. Descriptive Statistics of Dataset

Application Domain	Requirements in Train-Set	Requirements in Test-Set	Total Requirements
Energy	532	94	626
Entertainment	400	71	471
Health	504	89	593
Safety	758	134	892
Other	327	57	384

C. Sector-wise most frequent words

Domain	Top 14 most frequent words
Energy	home, smart, food, time, know, clean, automatically, house, water, temperature, air, health, need, pet
Entertainment	energy, home, save, smart, turn, room, light, water, temperature, automatically, electricity, house, time, money
Health	home, music, smart, room, time, tv, play, turn, house, voice, automatically, watch, favorite, movie
Safety	home, door, smart, house, know, alert, safe, child, lock, alarm, automatically, open, window, pet
Other	home, smart, time, automatically, door, water, know, house, need, open, food, dog, day, turn

D. Additional stop-word list:

home	smart	time	automatically	house	know	water
turn	temperature	shower	alert	dog	room	phone

E. Word clouds for each domain (after removal of stop-words and additional stop-words)

Fig 1. (a) Energy

Fig 1. (b) Entertainment

Fig 1. (c) Health

Fig 1. (d) Safety

Fig 1. (e) Safety

II. Implementation

A. Approach 1 (Implemented with Sklearn)

(a) Importing required libraries.

```
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from scipy.stats import randint as sp_randint
from sklearn.feature_selection import SelectKBest, chi2, f_classif, mutual_info_classif
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
import pickle
from sklearn.svm import SVC
```

(b) Define grid to be searched over on the pipeline.

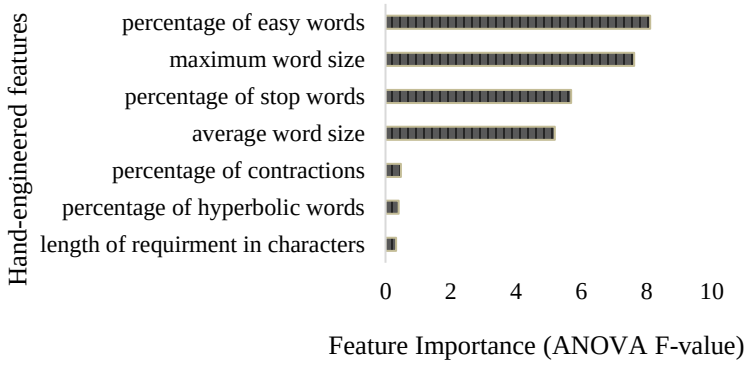
```
def tuning(Model,train_f,ytrain):
  selection = SelectKBest()
  start,end = (int)(train_f.shape[1]/2),train_f.shape[1]
  if Model == 'SVC': model = SVC(random_state=2020)
  elif Model == 'RF': model = RandomForestClassifier(random_state=2020)
  elif Model == 'KNN': model = KNeighborsClassifier()
  elif Model == 'MLP': model = MLPClassifier(activation='relu',solver='adam',random_state=2020)
  elif Model == 'XGB': model = XGBClassifier(activation='relu',solver='adam',random_state=2020)
  pipeline = Pipeline([("features", selection), (Model, model)])

  if Model == 'SVC':
    param_grid = dict(features__k=np.arange(start,end,1),
                      features__score_func=[f_classif],
                      SVC__C=[0.1, 1, 10, 100, 1000],
                      SXC__kernel=['linear','poly','rbf'],
                      SVC__gamma=[1, 0.1, 0.01, 0.001, 0.0001]
                      )
  elif Model == 'RF':
    param_grid = dict(features__k=np.arange(start,end,1),
                      features__score_func=[f_classif],
                      RF__max_features = ['auto', 'sqrt'],
                      RF__n_estimators = [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000],
                      RF__max_depth = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
                     )
  elif Model == 'KNN':
    #print(model.get_params)
    param_grid = dict(features__k=np.arange(start,end,1),
                      features__score_func=[f_classif],
                      KNN__n_neighbors =range(1, 100, 2),
                      metric=['euclidean','cosine']
                     )
  elif Model == 'MLP':
    param_grid = dict(features__k=np.arange(start,end,1),
                      features__score_func=[f_classif],
                      MLP__hidden_layer_sizes = [(5,),(10,),(15,),(20,),(25,),(30,),(40,),(50,),(70,),(80,),(90,),(100,)],
                      MLP__alpha = [0.00005,0.0005,0.001,0.005,0.01,0.1,0.5,0.6,0.7,0.8,0.9])
  elif Model == 'XGB':
    param_grid = dict(features__k=np.arange(start,end,1),
                      features__score_func=[f_classif],
                      XGB__max_depth=range(3,10,2),
                      XGB__gamma=[i/10.0 for i in range(0,5)],
                      XGB__learning_rate = [0.05, 0.10, 0.25],
                      XGB_lambda=[0.001,0.1,1]
                     )
```

```
grid_search = GridSearchCV(pipeline, param_grid=param_grid, cv=10, verbose=10,scoring='accuracy')
grid_search.fit(train_f, ytrain)
return grid_search.best_estimator_
```

(c) Feature Importance for Hand-engineered features

Feature	ANOVA F-value
length of requirement in characters	0.32879427
percentage of hyperbolic words	0.40527909
percentage of contractions	0.47148293
average word size	5.19404783
percentage of stop words	5.69315744
maximum word size	7.62893018
percentage of easy words	8.11528589



B. Approach 2 (Implemented with Keras)

(a) LSTM model architecture used in approach 2, with Adam optimizer.

```
def LSTM_tf(vocabulary_size,embed_size,lr,embedding_matrix,hidden_dim,output_size,seq_length):
  opt=keras.optimizers.Adam(learning_rate=lr)
  tf.random.set_seed(10)
  model_lstm = Sequential()
  if embedding_matrix==None:
    model_lstm.add(Embedding(vocabulary_size, embed_size, input_length=seq_length,trainable=True))
  else:
    model_lstm.add(Embedding(vocabulary_size, embed_size, input_length=seq_length,trainable=True,embeddings_initializer=tf.keras.initializers.Constant(embedding_matrix)))
  model_lstm.add(Bidirectional(LSTM(hidden_dim)))
  model_lstm.add(Dense(output_size, activation='softmax'))
  model_lstm.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
  return model_lstm
```

(b) Selected hyperparameters after grid search (with pre-trained glove embedding)

```
input_dic={
    'vocabulary_size':len(tok.word_index)+1,
    'embed_size':100,
    'lr'=3e-4
    'embedding_matrix':embed_matrix,
    'hidden_dim':16,
    'output_size':5,
    'seq_length':xTrain_pad.shape[1]
}
```

Note: 1. In case of random initialization we do not argument an embedding matrix.
2. We use the [1] architecture for CNN implementation.

C. Approach 3 (Implemented with Pytorch)

(a) Loading Huggingface's implementation

```
from transformers import BertTokenizer, RobertaTokenizer

# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
tokenizer = RobertaTokenizer.from_pretrained('roberta-base', do_lower_case=True)

```

(b) Processing the requirements before fine-tuning the transformer model

```
# Tokenize all of the sentences and map the tokens to thier word IDs.
input_ids = []
attention_masks = []

# For every sentence...
for sent in sentences:
    # `encode_plus` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = 128,           # Pad & truncate all sentences.
                        pad_to_max_length = True,
                        truncation=True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt'    # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_dict['input_ids'])

    # And its attention mask (simply differentiates padding from non-padding).
    attention_masks.append(encoded_dict['attention_mask'])

```

(c) Optimizer and scheduler for training

```
 # Note: AdamW is a class from the huggingface library (as opposed to pytorch)
optimizer = AdamW(model.parameters(),
                  lr = 2e-5, # args.learning_rate
                  eps = 1e-8 # args.adam_epsilon
                )

from transformers import get_linear_schedule_with_warmup

epochs = 4

# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0, # Default value in run_glue.py
                                            num_training_steps = total_steps)
```

III. Experimental Evaluation

A. Results for all three approaches.

(a) Approach 1

Model	Sentence Encoding	k	Precision	Recall	Fscore
Random Forest	Averaged Glove + features	101	64.29	67.19	63.21
	TF-IDF Weighted Glove + features	95	61.6	64.26	62.05
	Universal Sentence Encoding	330	**70.01**	**71.68**	69.71
XGBoost	Averaged Glove + features	93	65.15	66.74	65.63
	TF-IDF Weighted Glove + features	104	65.07	67.19	65.48
	Universal Sentence Encoding	341	68.64	70.33	68.87
SVM	Averaged Glove + features	83	68.3	70.78	69.11
	TF-IDF Weighted Glove + features	101	67.71	69.88	68.2
	Universal Sentence Encoding	519	68.42	71.46	69.44
KNN	Averaged Glove + features	59	63.8	66.29	63.96
	TF-IDF Weighted Glove + features	72	62.52	65.39	61.73
	Universal Sentence Encoding	384	67.99	70.11	67.8
MLP	Averaged Glove + features	59	66.97	68.31	67.33
	TF-IDF Weighted Glove + features	72	63.34	65.39	63.3
	Universal Sentence Encoding	519	69.9	71.46	**70.46**

(b) Approach 2

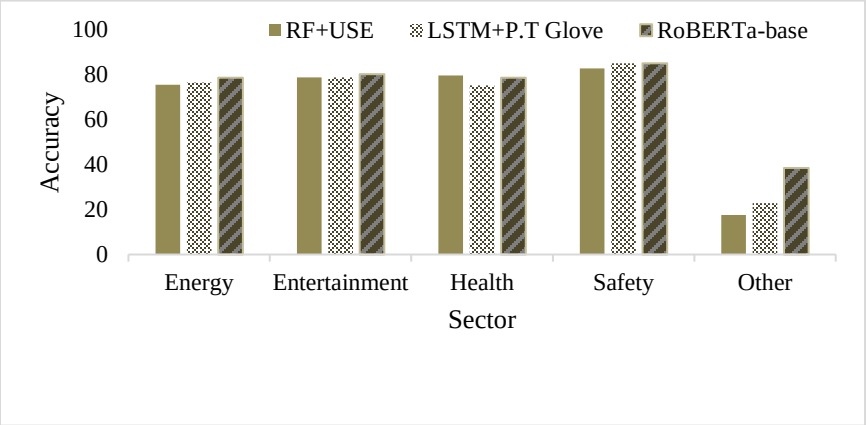
Model	Precision	Recall	F-score
CNN	66.45	68.31`	65.07
CNN+P.T Glove	68.77	70.11	67.75
Bi-LSTM	68.02	70.33	68.54
Bi-LSTM+P.T Glove	**71.39**	**72.36**	**70.81**

(c) Approach 3

Model	Precision	Recall	F-score
BERT-base-uncased	72.25	74.15	72.14
DistilRoBERTa-base	73.69	75.06	73.55
RoBERTa-base	**75.56**	**75.73**	**75.13**

B. Sector-wise Accuracy of the best model in each approach

Model	Energy	Entertainment	Health	Safety	Other
RF+USE	75.53	78.87	79.77	82.83	17.54
LSTM+P.T Glove	76.59	78.87	75.20	85.07	22.80
RoBERTa-base	78.72	80.28	78.65	85.07	38.59



References

[1] Yoon Kim. 2014. Convolutional neural networks for sentence classification. arXiv preprint arXiv:1408.5882 (2014).