SI 618 Team Project Report

AIGC Prompts Category Classifier

Team members: *Yifan Li (ivanlyf) section 001; Yan Lu (yanlunar) section 001;*

1. Motivation

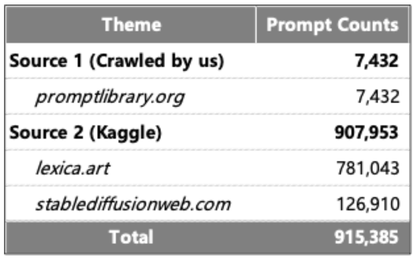
When using the AIGC (Artificial Intelligence Generated Content) platform, crafting effective prompts is a critical step in generating high-quality content. Prompts serve as the input or guidance for the AI model to generate meaningful responses or content. Good prompts are clear and structured instructions, questions, or cues provided to AI systems and can lead to better production of the generated content.

However, even though there are hundreds of AIGC models and application tools for generating text, images, and even videos (e.g. GPT, Llama, Stable Diffusion, Midjourney…), online guidance for producing high-quality prompts is few. Website builders may find it hard to find appropriate and high-quality prompts based on their needs in a certain category.

Here, we propose a classifier to categorize AIGC prompts, which can correctly and precisely classify prompts into appropriate categories to make it easier for people to refer to.

2. Data Sources

We have two data sources in total, and prompts from these two datasets are from different sources. The first is crawled by ourselves while the second comes from Kaggle. An overview of the data source is provided in Fig. 2.1. The purpose of combining two datasets from different sources is to increase the diversity and variety of our data, which will be helpful in the model training process since it increases the richness of the training set as well as strengthens the models' generalization ability.



Theme	Prompt Counts
Source 1 (Crawled by us)	**7,432**
promptlibrary.org	7,432
Source 2 (Kaggle)	**907,953**
lexica.art	781,043
stablediffusionweb.com	126,910
Total	**915,385**

Figure 2.1: Description of Data Sources

Data source 2 is utilized to generate tags for prompts in data source 1, which serves as features for future modeling as shown in Fig. 2.2. We first find the top 100 words with the highest frequency from data source 2 and then reserve those words that also exist in the prompts from data source 1. Finally, we find 27 words in common and save them as tags (which will be encoded as One-Hot).

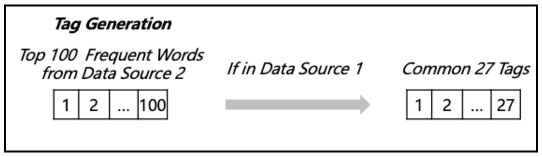
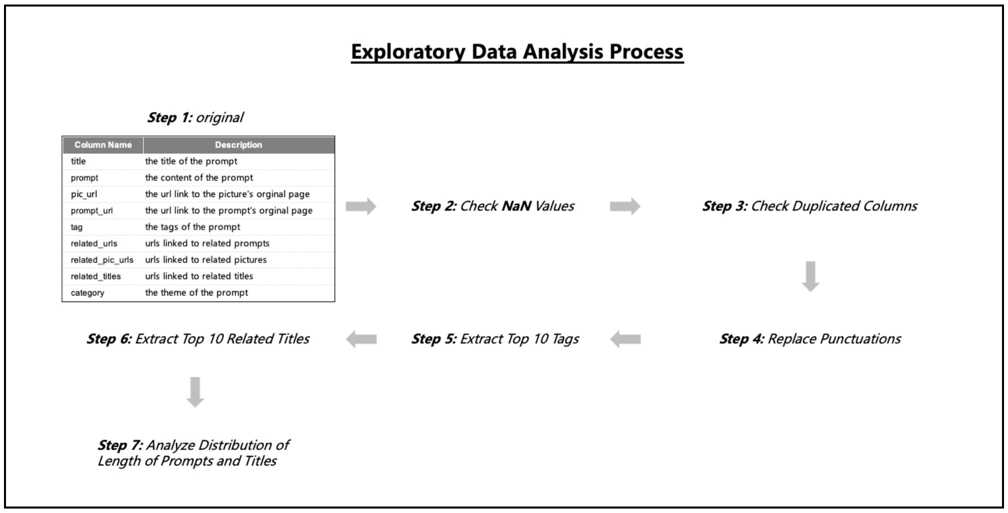


Figure 2.2: Data Source Join Process

3. Data Manipulation Methods

3.1 Exploratory Data Analysis (EDA)

The process of exploratory data analysis consists of seven distinct steps, as illustrated in Figure 3.1.1. The application of EDA facilitates the extraction of valuable information from the dataset, which can potentially serve as features in future modeling processes.


Exploratory Data Analysis Process
Step 1: original
Step 2: Check NaN Values
Step 3: Check Duplicated Columns
Step 4: Replace Punctuations
Step 5: Extract Top 10 Tags
Step 6: Extract Top 10 Related Titles
Step 7: Analyze Distribution of Length of Prompts and Titles

Column Name	Description
title	the title of the prompt
prompt	the content of the prompt
pic_url	the url link to the picture's orginal page
prompt_url	the url link to the prompt's orginal page
tag	the tags of the prompt
related_urls	urls linked to related prompts
related_pic_urls	urls linked to related pictures
related_titles	urls linked to related titles
category	the theme of the prompt

Figure 3.1.1: Exploratory Data Analysis Process

The results of our exploratory data analysis are presented in Figure 3.1.2. We observe that the majority of items fall under the "category nature," followed by "illustration," and then "digital_art" and "design." It is noteworthy that the top four tags align with the category names. However, it's essential to note that having a tag that matches the category name does not guarantee automatic categorization. The categorization process takes multiple factors into consideration, such as the context of the prompts and the presence of other tags.

Furthermore, when examining related titles, we observe that four titles exhibit strong associations with others. This suggests that a prompt's similarity to these titles could serve as a valuable feature. Regarding prompt length, the distribution is skewed to the right, indicating that there is a higher concentration of data on the left side of the distribution and a relatively long tail above the average value. For the title length, we can see that most titles have length 4 and the majority of titles have length 3-5.

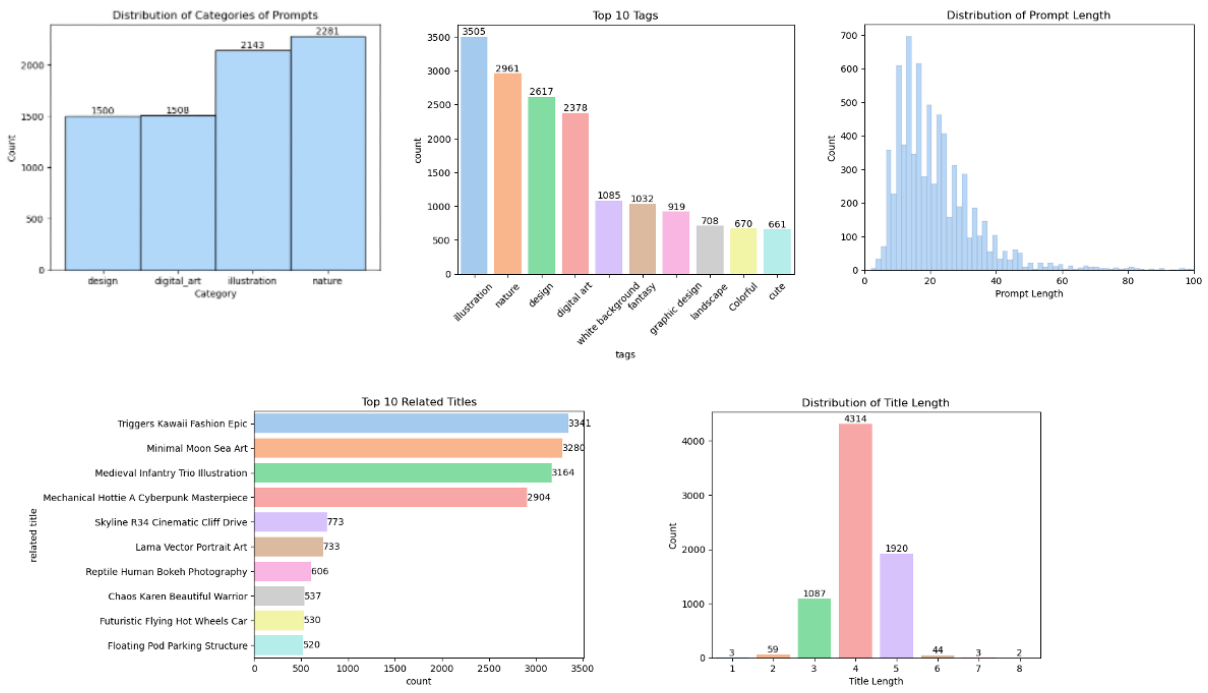
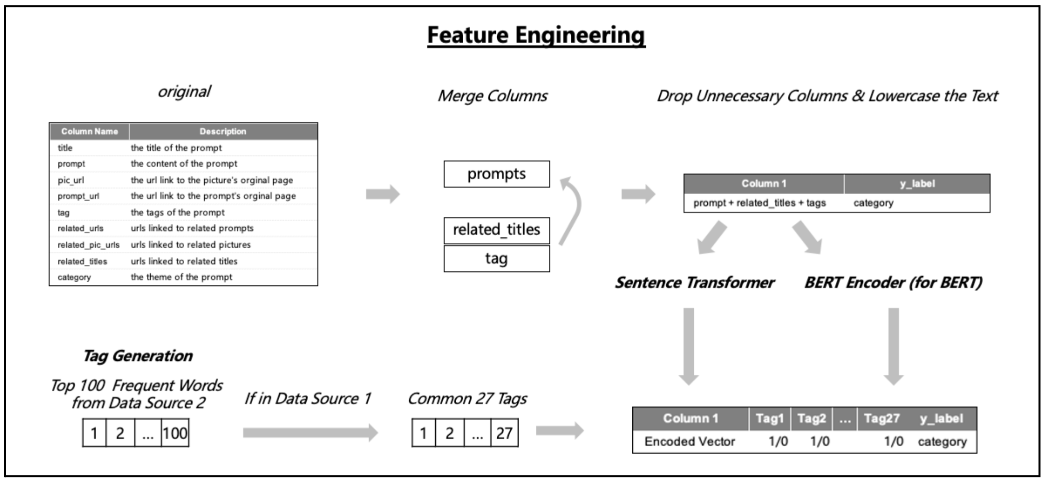


Figure 3.1.2: Results of EDA

3.2 Feature Engineering

The process of feature engineering is illustrated in Fig. 3.2.1. We first merge *related_titles* and *tag* into prompts, and then drop unnecessary columns as well as lowercase the text. For BERT, we utilize BERT Encoder to get the embedding text, while for others we use sentence transformer to play the same role. The details of the encoder will be introduced later. Meanwhile, as mentioned before, we use data source 2 to generate the tags and retain those that also exist in data source 1. By integrating the 27-column metric with the transformed dataset, all necessary data are ready for modeling.



Column Name	Description
title	the title of the prompt
prompt	the content of the prompt
pic_url	the url link to the picture's orginal page
prompt_url	the url link to the prompt's orginal page
tag	the tags of the prompt
related_urls	urls linked to related prompts
related_pic_urls	urls linked to related pictures
related_titles	urls linked to related titles
category	the theme of the prompt

Column 1	y_label
prompt + related_titles + tags	category

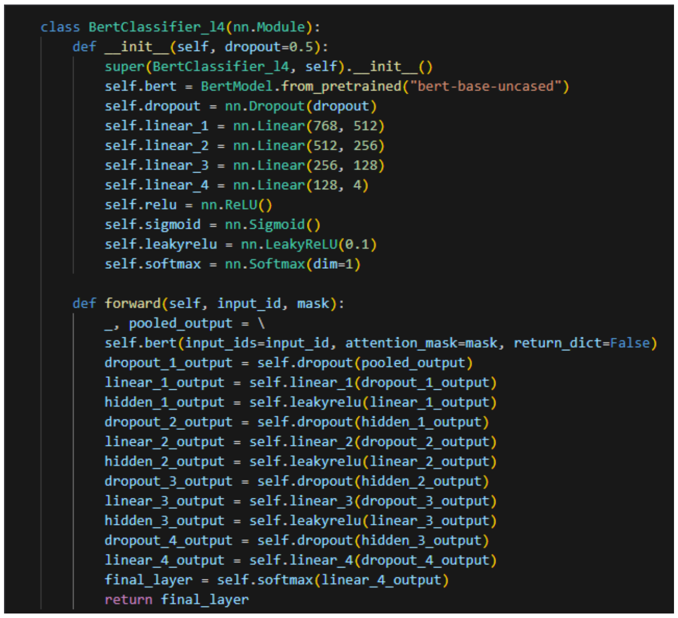
Column 1	Tag1	Tag2	...	Tag27	y_label
Encoded Vector	1/0	1/0		1/0	category

Figure 3.2.1: Percentage of Categories for Modeling

4. Modeling & Analysis

Now that we had all the data needed, we wanted to find out whether it was possible to find the relationship between prompts and their categories. The first step was to interpret information like prompts and tags into vectors or embeddings that the machine could understand. To achieve this, we used two approaches: the first one is using a sentence transformer that would map the sentence to a 384-dimensional vector space (https://huggingface.co/sentence-transformers/msmarco-MiniLM-L12-cos-v5), and the second one is a more state-of-art model called BERT (Bidirectional Encoder Representations) which also mapped sentences to vectors but it took relationships between words into account, and thus achieving a theoretically better representation of the whole sentence (https://huggingface.co/bert-base-uncased). To complete our classification task, we utilized both of the approaches and compared their differences.

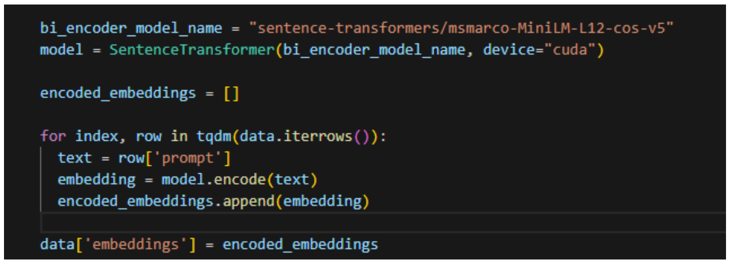
Based on these two encoders, we thus created two types of models - one is BERT + neural networks that define the downstream task of classification, while another one is sentence transformer + traditional classification models like Random Forest, KNN, etc. For BERT, we've tested network structures with different number and types of layers. After comparison, the selected BERT model structure is as follows:



```python
class BertClassifier_l4(nn.Module):
    def __init__(self, dropout=0.5):
        super(BertClassifier_l4, self).__init__()
        self.bert = BertModel.from_pretrained("bert-base-uncased")
        self.dropout = nn.Dropout(dropout)
        self.linear_1 = nn.Linear(768, 512)
        self.linear_2 = nn.Linear(512, 256)
        self.linear_3 = nn.Linear(256, 128)
        self.linear_4 = nn.Linear(128, 4)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.leakyrelu = nn.LeakyReLU(0.1)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, input_id, mask):
        _, pooled_output = \
        self.bert(input_ids=input_id, attention_mask=mask, return_dict=False)
        dropout_1_output = self.dropout(pooled_output)
        linear_1_output = self.linear_1(dropout_1_output)
        hidden_1_output = self.leakyrelu(linear_1_output)
        dropout_2_output = self.dropout(hidden_1_output)
        linear_2_output = self.linear_2(dropout_2_output)
        hidden_2_output = self.leakyrelu(linear_2_output)
        dropout_3_output = self.dropout(hidden_2_output)
        linear_3_output = self.linear_3(dropout_3_output)
        hidden_3_output = self.leakyrelu(linear_3_output)
        dropout_4_output = self.dropout(hidden_3_output)
        linear_4_output = self.linear_4(dropout_4_output)
        final_layer = self.softmax(linear_4_output)
        return final_layer
```

Figure 4.1: BERT Network Structure

For the second type of model, embedding is realized by the following code:

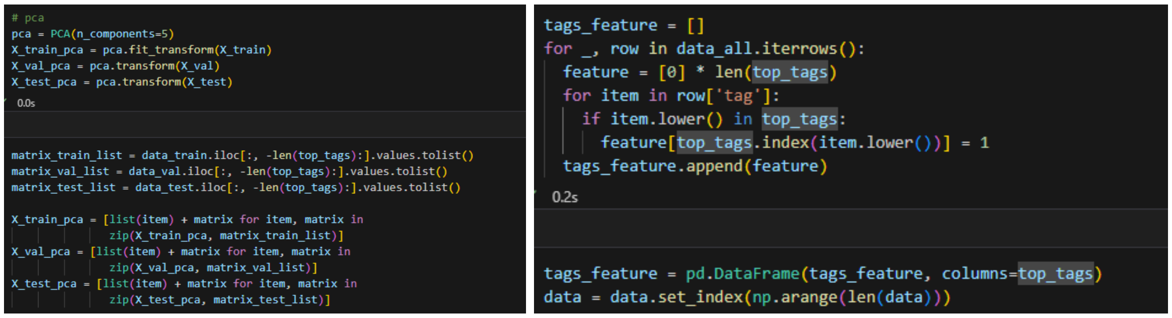


```python
bi_encoder_model_name = "sentence-transformers/msmarco-MiniLM-L12-cos-v5"
model = SentenceTransformer(bi_encoder_model_name, device="cuda")

encoded_embeddings = []

for index, row in tqdm(data.iterrows()):
  text = row['prompt']
  embedding = model.encode(text)
  encoded_embeddings.append(embedding)

data['embeddings'] = encoded_embeddings
```

Figure 4.2: Sentence Transformer Encoding

After encoding, the next step was to prepare models to realize the classification task. However, though BERT + neural network performed generally well during the training phase, we met effectiveness and efficiency problems when trying to fit the embeddings with Random Forest directly. As also shown in Table 4.1, when using the Random Forest without other information and decomposition techniques, the model not only ran quite slowly during training step, but also had poor performance - the overall accuracy was only 0.61 and the recall rate for category 'design' was only 0.27, which we considered far from satisfaction.

Therefore, we tried to improve our models in terms of accuracy and efficiency. The approach we used was to include tag information in our data set. To be specific, we wanted to utilize the second data set to find out what might be the common features across prompts. The idea was to calculate the top 100 words in the prompts of the second data set and the top 100 common tags in the first data set. Then combine them together to get the intersections, which turns out to be 27 tags. We encoded them to be 27 features. For each of these 27 features, it would be 1 if the prompt had the corresponding tag, otherwise it would be 0. Besides adding additional features, we also did matrix decomposition operations on the original 384-dimensional embeddings. We used PCA to select 5 features from the embeddings and concat them with tag information to be our new feature matrix.



```
# pca
pca = PCA(n_components=5)
X_train_pca = pca.fit_transform(X_train)
X_val_pca = pca.transform(X_val)
X_test_pca = pca.transform(X_test)
 0.0s

matrix_train_list = data_train.iloc[:, -len(top_tags):].values.tolist()
matrix_val_list = data_val.iloc[:, -len(top_tags):].values.tolist()
matrix_test_list = data_test.iloc[:, -len(top_tags):].values.tolist()

X_train_pca = [list(item) + matrix for item, matrix in
               zip(X_train_pca, matrix_train_list)]
X_val_pca = [list(item) + matrix for item, matrix in
               zip(X_val_pca, matrix_val_list)]
X_test_pca = [list(item) + matrix for item, matrix in
               zip(X_test_pca, matrix_test_list)]
```

```
tags_feature = []
for _, row in data_all.iterrows():
  feature = [0] * len(top_tags)
  for item in row['tag']:
    if item.lower() in top_tags:
      feature[top_tags.index(item.lower())] = 1
  tags_feature.append(feature)
 0.2s

tags_feature = pd.DataFrame(tags_feature, columns=top_tags)
data = data.set_index(np.arange(len(data)))
```

Figure 4.3: Matrix Decomposition and Feature Addition

With all data preparation work done, we finally proceeded to our model training and fine-tuning part, we in total prepared seven models to solve our classification task. The first one is Random Forest without tag information and PCA decomposition. The next five are models with tag information and PCA decomposition: Random Forest, KNN, SVM, XGBoost, and Gradient Boosting. And the final one is BERT (Bidirectional Encoder Representations), combined with a neural network to realize the classification function. Generally, for the traditional classification models, we performed GridSearch to get the best parameters and fine-tuned it on the validation set. The overall performance of our final models on the validation set are listed in Table 4.1 and is visualized in Figure 4.4. From this table, we can see that among the traditional models, Random Forest and SVM with tag information seem to have the best average performance on the validation set, and other models with tag information can all achieve an average accuracy over 61%. The original Random Forest was poor in performance since it had no information about related tags of the prompts.

Table 4.1: Accuracy of Different Models on Validation Set

Model	**Random Forest(No Tags)**	**Random Forest**	**KNN**	**SVM**	**XGBoost**	**Gradient Boosting**	**BERT**
Accuracy	0.571928	0.66764	0.621802	0.663532	0.615153	0.62591	0.78292

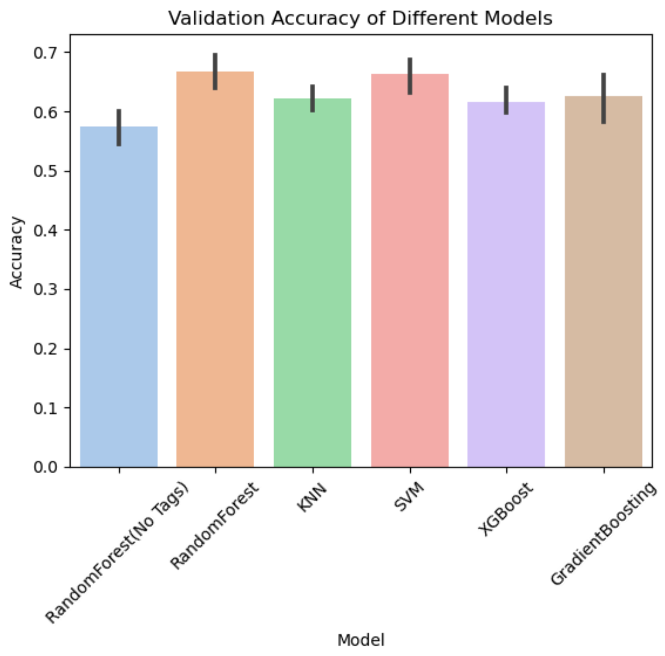


Figure 4.4: Validation Accuracy of Different Models

After training and fine-tuning our models, we tested the performance of our models on the test set. Metrics like precision, recall, F1-score and accuracy were used to do the evaluation. It turns out that on the test set, traditional classification models didn't have better performance than our newly-introduced model BERT. While models like Random Forest, KNN and SVM could achieve accuracy of around 0.72, they performed badly at the recall rate. For most models, the recall rates were below 0.35 for category 1. However, BERT as a more comprehensive and complicated model, could achieve accuracy of 80% while remaining the recall rate to be greater than 0.5. This shows the strength and advantage of this state-of-art model in our classification task. The detailed statistics are listed in the following tables as well as Figure 4.5.

Table 4.2: Random Forest without Tag Information and PCA

	precision	recall	f1-score	support
0	0.59	0.27	0.37	136
1	0.7	0.32	0.44	155
2	0.6	0.81	0.69	214
3	0.6	0.8	0.69	239
accuracy	0.61			744

Table 4.3: Random Forest with Tag Information and PCA

	precision	recall	f1-score	support
0	0.7	0.73	0.71	136
1	0.89	0.22	0.35	155
2	0.66	0.95	0.78	214

3	0.77	0.82	0.8	239
accuracy		0.72		744

Table 4.4: KNN with Tag Information and PCA

	precision	recall	f1-score	support
0	0.73	0.7	0.71	136
1	0.79	0.24	0.37	155
2	0.66	0.99	0.79	214
3	0.77	0.8	0.78	239
accuracy		0.72		744

Table 4.5: SVM with Tag Information and PCA

	precision	recall	f1-score	support
0	0.7	0.72	0.71	136
1	0.76	0.31	0.44	155
2	0.66	0.96	0.79	214
3	0.81	0.78	0.79	239
accuracy		0.72		744

Table 4.6: XGBoost with Tag Information and PCA

	precision	recall	f1-score	support
0	0.52	0.62	0.56	136
1	0.46	0.34	0.39	155
2	0.59	0.64	0.61	214
3	0.75	0.74	0.75	239
accuracy		0.6		744

Table 4.7: Gradient Boosting with Tag Information and PCA

	precision	recall	f1-score	support
0	0.63	0.76	0.69	136
1	0.72	0.32	0.45	155
2	0.65	0.84	0.73	214
3	0.81	0.78	0.79	239
accuracy		0.7		744

Table 4.8: BERT + Neural Network

	precision	recall	f1-score	support
0	0.85	0.57	0.68	136
1	0.9	0.6	0.72	155
2	0.66	0.97	0.79	214
3	0.91	0.9	0.9	239
accuracy	0.8			744

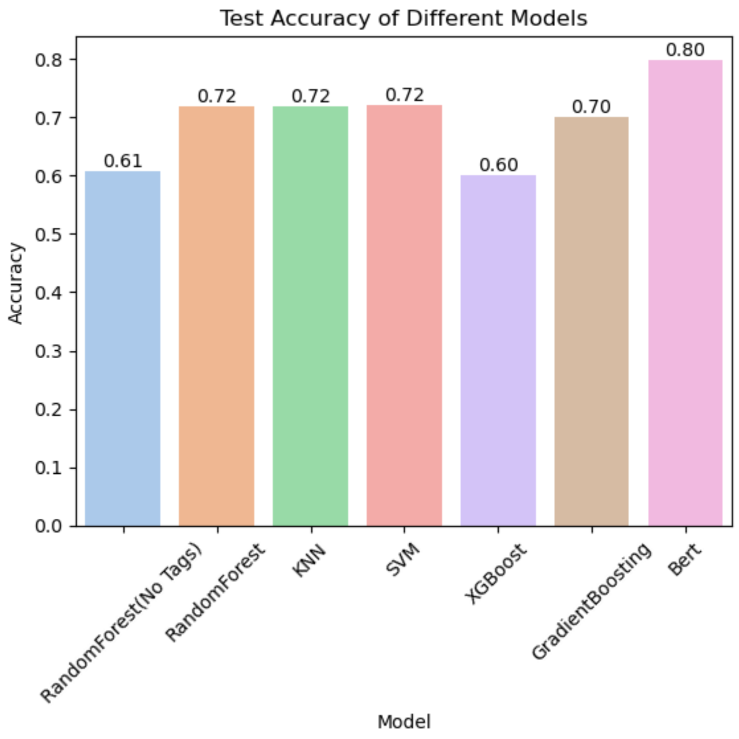


Figure 4.5: Test Accuracy of Different Models

In conclusion, one interesting insight that we gained through our analysis was that the related tags of a prompt did have a huge impact on improving the accuracy of predicting categories, and also using PCA to decompose the data was necessary when we had too many features. The final result on the test set shows that BERT might be the best model for high accuracy and recall rate, but considering its relatively longer training process and complexity, traditional models still have their strength of speed and explainability. It is promising to use either of these kinds of models for companies or organizations that may involve work on classification of AIGC prompts.

5. Statement of Work

Yifan Li: Yifan Li assumes a central role in setting up the evaluation pipeline and refining the data/model preparation process. He is also responsible for authoring section 1, 3, 4 of the final report.

Yan Lu: Yan Lu spearheads the setup of the training pipeline, including vital tasks such as data cleaning, feature extracting and model preparation. He is also responsible for authoring section 1, 2, 3 of the final report.

6. Reference

Kajal Kumari, Building Language Models: A Step-by-Step BERT Implementation Guide. https://www.analyticsvidhya.com/blog/2023/06/step-by-step-bert-implementation-guide/. Accessed 7 Dec. 2023.