NLP

Overview

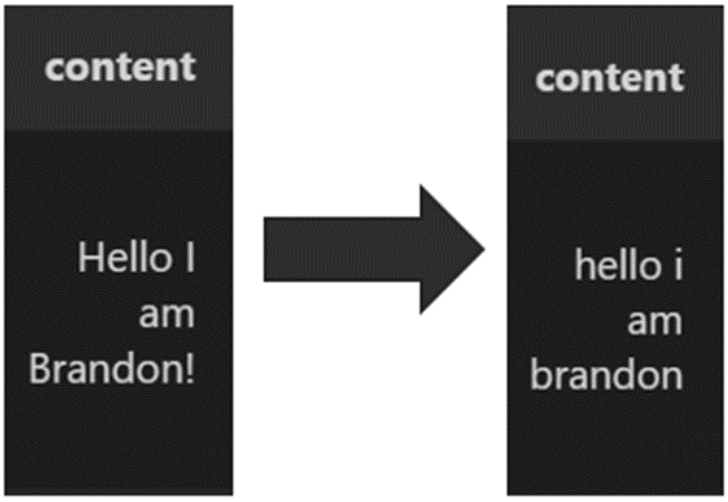
Natural Language Processing (NLP) is a field of programming concerned with extracting usable numeric data or other important information from human language or text.

When it comes to the Nephrotex Virtual Internships dataset, the actual chat or ‘content’ column is the primary source of information which determined other features such as ‘m_making_design_choices’ or ‘j_communication’, which are Boolean columns which denote the purpose or function of a chat message. As such, to introduce any new features or information to the dataset, they would have to be extracted from the actual chat transcripts using NLP techniques.

For this dataset, TF-IDF will be used to find the most important words in all the chat data and use them to establish new features which may have some stronger and more relevant correlations to the outcome score.

Text-Data Pre-Processing

Before we apply any NLP techniques to the data, the text data needs to undergo some transformation to make the TF-IDF more effective. Most NLP techniques, including the one intended to be used on this dataset, involve tokenizing the text by words or splitting bodies of text by individual words. However, features of language things such as capital letters or standard features of punctuation, mean that the tokens ‘Mean’, ‘mean.’ and ‘mean’ are recognized as different words. To avoid any inaccuracies, the chat data was stripped of all punctuation and set to lower case only.



TF-IDF

Term frequency-inverse document frequency (TF-IDF) is an NLP technique used to find the importance of each word in a document or series of documents, in relation to those documents. This is achieved by finding the number of times a certain term appears in

one document (the term frequency) and multiplying it by the proportion of documents in the set which contain that (inverse document frequency), this process usually returns a score between 0 and 1, where the higher the score the more relevant, or important, the word can be considered. Whilst the TF helps to recognize words with high usage, the IDF helps balance this frequency count by returning lower scores for those words which appear in more documents, to avoid confusion caused by standard lexical features of the English language, e.g., ‘the’, ‘is’, ‘I’, etc.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

The formula used to calculate TF-IDF

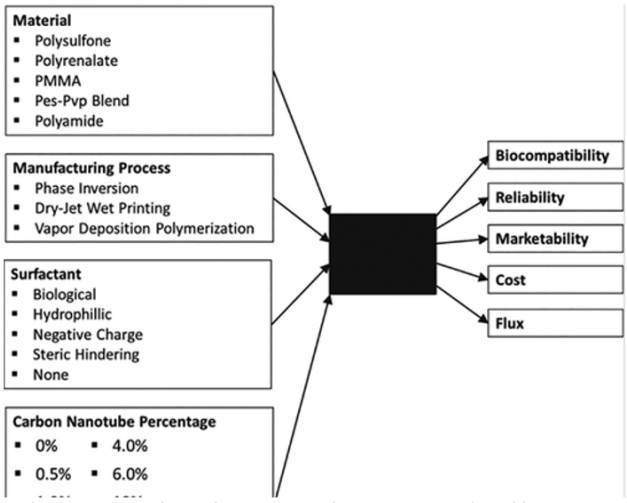
Implementing TF-IDF on the Virtual Internships data involved utilising the scikit learn package ‘TfidfVectorizer’. When initialising the vectorizer, stop-words were set to ‘English’ so that basic structural words in the English language would be ignored by the vectorizer. The vectorizer was also initialised to return a list of 25 words with highest TF-IDF scores, using the ‘max_features’ parameter, this was aimed to make initial analysis a bit easier and prevent over-analysis of the TF-IDF results in-order to avoid confusion.

```
Feature Names n ['agree' 'best' 'blood' 'cnt' 'cost' 'did' 'flux' 'good' 'hindering' 'im'
 'just' 'like' 'make' 'marketability' 'negative' 'notebook' 'prototype'
 'prototypes' 'reactivity' 'reliability' 'steric' 'surfactant' 'think'
 'yeah' 'yes']
```

List of the 25 most important words according to the TfidfVectorizer.

New Features

Though there are now 25 possible words from which new features could be designed it is important to note that the issue of relevancy had to be considered. Making features without any reason would cause inaccuracies in any later modelling and analyses.



A diagram describing the inputs and outputs considered by participants in the Virtual Internship program.

Upon some further research about Nephrotex, it was learned that some components of the virtual internship task could be categorised as either inputs or outputs. Using the diagram seen to the side and the list extracted from the TfidfVectorizer it was possible to construct two new, contextually relevant, features for the dataset. Both of which counted the following words in each chat message, producing a sum of the number of input and output terms contained in one chat entry. The hypothesis being the more a participant used input and output terms the more likely they were having relevant conversations which would lead to higher outcome scores.

Input:

- Surfactant
- Steric
- Hindering
- CNT (Carbon Nanotubes)

Output:

- Reliability
- Cost
- Flux
- Marketability

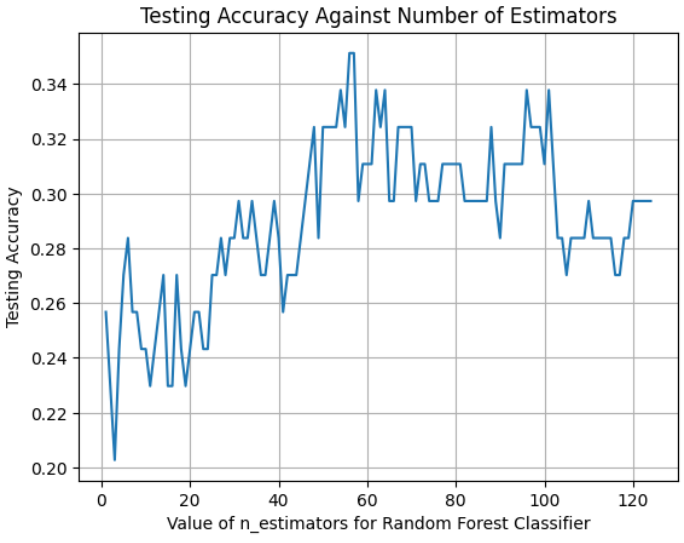
Modelling with New Features

```
m_experimental_testing 0.078
m_making_design_choices 0.117
m_asking_questions 0.138
j_customer_consultants_requests 0.062
j_performance_parameters_requirements 0.093
j_communication 0.046
wordCount 0.195
input 0.136
output 0.134
```

To begin exploring the effects of the new features, a user-based model was constructed using Random Forest Classifier. This model produced a 0.297 score for accuracy at an optimal number of estimators of 69, which was an improvement from the previous user model's 0.27, however it was not substantial enough to be deemed relevant.

Though there was no evident improvement in accuracy due to addition of new features, when taking a look at the feature importances for the model, both 'input' and 'output' are the most important features after word count. This was an indication that creating the new features were a step in the right direction when it came to constructing better models.

Taking the feature importances into consideration a new model using RFC was made using just 'output', 'input', 'wordCount', 'm_asking_questions' and 'm_making_design_choices', the features with importance greater than 0.1.



[319]:
```
knn = KNeighborsClassifier(n_neighbors = 9).fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
metrics.accuracy_score(Y_test, Y_pred)
```

[319]:
```
0.32432432432432434
```

5 Importance of Certain Words

[320]:
```
new_df = clean_df
#Dropping NAs and users
new_df = new_df.dropna()
new_df = new_df[new_df.RoleName == 'Mentor']
#Adding column for each group
new_df['Group'] = new_df['group_id'].astype(str) + new_df['implementation']
```

[321]:
```
without_dups = new_df.drop_duplicates(subset=['userIDs'])
group_nums = without_dups.sort_values(by = ['Group'])
#group_nums = without_dups.groupby(['Group'], as_index = False)
```

[322]:
```
group_nums
```

[322]:
```
Empty DataFrame
Columns: [Unnamed: 0, content, RoleName, userIDs, implementation, Line_ID,
ChatGroup, group_id, roomName, m_experimental_testing, m_making_design_choices,
m_asking_questions, j_customer_consultants_requests,
j_performance_parameters_requirements, j_communication, OutcomeScore, wordCount,
Group]
Index: []
```

[323]:
```
without_dups
```

[323]:
```
Empty DataFrame
Columns: [Unnamed: 0, content, RoleName, userIDs, implementation, Line_ID,
ChatGroup, group_id, roomName, m_experimental_testing, m_making_design_choices,
m_asking_questions, j_customer_consultants_requests,
j_performance_parameters_requirements, j_communication, OutcomeScore, wordCount,
Group]
Index: []
```

6 Finding The Sum of Certain Words

6.1 By Team

6.1.1 Sorting

First we need to find the number of people in each team to calculate the average score for each team.

This gets rid of the duplicate users for each group and then the count for any column is the amount of users in that group.

[324]:
```
new_df = clean_df
#Dropping NAs and mentors
new_df = new_df.dropna()
new_df = new_df[new_df.RoleName != 'Mentor']
#Adding column for each group
new_df['Group'] = new_df['implementation'] + new_df['group_id'].astype(str)

without_user_dups = new_df.drop_duplicates(subset=['userIDs'])
dupless_group_nums = without_user_dups.groupby(['Group'], as_index = False)

sum_table = dupless_group_nums.sum()
count_table = dupless_group_nums.count()
sum_table['outcome_per_student'] = sum_table['OutcomeScore'] /␣
 ↪count_table['userIDs']
sum_table['group_id'] = count_table['group_id']
sum_table[['outcome_per_student']]
```

[324]:
```
    outcome_per_student
0              3.571429
1              3.666667
2              2.571429
3              2.500000
4              2.166667
..                  ...
70             2.200000
71             4.400000
72             4.800000
73             4.600000
74             5.500000

[75 rows x 1 columns]
```

[325]:
```
#Grouping by teams
team_group = new_df.groupby(['Group'], as_index = False)
#Getting the sum of the contributions
team_sum = team_group.sum()
#Adding the outcome per student
```

```
team_sum[['Average Score']] = sum_table[['outcome_per_student']]
```

[326]:
```
team_sum
```

[326]:
```
   Group  Unnamed: 0  userIDs  Line_ID  group_id  m_experimental_testing  \
0     a2       55756     1515    55756       598                      10
1     a3       79134     1893    79134       510                       6
2     a4      200807     4879   200807      1096                      25
3     a5      193183     4902   193183       960                       2
4     a6      340114     8465   340114      1602                       2
..   ...         ...      ...      ...       ...                     ...
70    o2     6216397   129357  6217447       700                      10
71    o3     5305801   109842  5306680       879                       6
72    o4     3676635    75978  3677235       800                      11
73    o5     6314090   130116  6315104      1690                      16
74    o6     5554645   114095  5555521      1752                      16

    m_making_design_choices  m_asking_questions  \
0                        21                  54
1                        16                  25
2                        25                  59
3                        32                  24
4                        34                  40
..                      ...                 ...
70                       31                  41
71                       26                  29
72                       21                  18
73                       36                  59
74                       28                  36

    j_customer_consultants_requests  j_performance_parameters_requirements  \
0                                 2                                     21
1                                 3                                     11
2                                 2                                     18
3                                 6                                     14
4                                 1                                     21
..                              ...                                    ...
70                                6                                     18
71                                4                                     19
72                                6                                     21
73                                6                                     11
74                                5                                     29

    j_communication  OutcomeScore  wordCount  Average Score
0                 1          1076       3007       3.571429
1                 2           619       2455       3.666667
2                 3           628       3234       2.571429
```

```
3                3          515       2470       2.500000
4                4          621       2827       2.166667
..             ...          ...        ...            ...
70               6          789       3956       2.200000
71               0         1192       3237       4.400000
72               3          972       2785       4.800000
73               5         1540       4484       4.600000
74               0         1549       3818       5.500000

[75 rows x 14 columns]
```

6.2 By User

6.2.1 Sorting

We need to find the amount of times certain words were said. We will be looking at some of the more common words found using nlp: 'surfactant' and 'steric.'

[327]:
```
new_df = clean_df
#Dropping NAs and mentors
new_df = new_df.dropna()
new_df = new_df[new_df.RoleName != 'Mentor']

#cleaning text/language data
new_df['content'] = new_df['content'].str.lower()
new_df['content'] = new_df['content'].str.translate(str.maketrans('', '', string.
 ↪punctuation))
#Adding column for each group
new_df['Group'] = new_df['implementation'] + new_df['group_id'].astype(str)
```

[328]:
```
new_df
```

[328]:
```
       Unnamed: 0             content RoleName  userIDs implementation  \
5               6  hello i am brandon   Player        2              a
6               7          i am zelin   Player        3              a
7               8                  hi   Player        3              a
8               9           i am jack   Player        4              a
9              10       hey im rachel   Player        5              a
...           ...                 ...      ...      ...            ...
19170      19174             exactly   Player      391              o
19171      19175         sounds good   Player      389              o
19172      19176                 yes   Player      392              o
19173      19177         sounds good   Player      388              o
19175      19179           precisely   Player      393              o

       Line_ID ChatGroup  group_id  \
5            6     PRNLT         2
6            7     PRNLT         2
```

```
7            8     PRNLT         2
8            9     PRNLT         2
9           10     PRNLT         2
...        ...       ...       ...
19170    19177    PESPVP         6
19171    19178    PESPVP         6
19172    19179    PESPVP         6
19173    19180    PESPVP         6
19175    19182    PESPVP         6

                                              roomName  \
5      Introduction and Workflow Tutorial with Entran...
6      Introduction and Workflow Tutorial with Entran...
7      Introduction and Workflow Tutorial with Entran...
8      Introduction and Workflow Tutorial with Entran...
9      Introduction and Workflow Tutorial with Entran...
...                                                  ...
19170  Reflection team discussion of first batch results
19171  Reflection team discussion of first batch results
19172  Reflection team discussion of first batch results
19173  Reflection team discussion of first batch results
19175  Reflection team discussion of first batch results

       m_experimental_testing  m_making_design_choices  m_asking_questions  \
5                           0                        0                   0
6                           0                        0                   0
7                           0                        0                   0
8                           0                        0                   0
9                           0                        0                   0
...                       ...                      ...                 ...
19170                       0                        0                   0
19171                       0                        0                   0
19172                       0                        0                   0
19173                       0                        0                   0
19175                       0                        0                   0

       j_customer_consultants_requests  j_performance_parameters_requirements  \
5                                    0                                      0
6                                    0                                      0
7                                    0                                      0
8                                    0                                      0
9                                    0                                      0
...                                ...                                    ...
19170                                0                                      0
19171                                0                                      0
19172                                0                                      0
19173                                0                                      0
```

```
19175                                0                                    0

       j_communication  OutcomeScore  wordCount Group
5                    0             4          4    a2
6                    0             4          3    a2
7                    0             4          1    a2
8                    0             4          3    a2
9                    0             2          3    a2
...                ...           ...        ...   ...
19170                0             5          1    o6
19171                0             7          2    o6
19172                0             5          1    o6
19173                0             8          2    o6
19175                0             4          1    o6

[16902 rows x 18 columns]
```

We can iterate through 16902 rows.

[329]:
```
#print(new_df[['content']].iloc[0])

for i in new_df[['content']].iloc[0]:
    print(i)
```

```
hello i am brandon
```

[330]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
doc_corpus = []
for i in new_df.index:
    doc_corpus.append(new_df['content'].loc[i])

# print(doc_corpus)

vec=TfidfVectorizer(stop_words='english', max_features= 25)
matrix=vec.fit_transform(doc_corpus)
# print("Feature Names n",vec.get_feature_names_out())
# print("Sparse Matrix n",matrix.shape,"n",matrix.toarray())

# print(matrix)
```

[331]:
```
word_count = []
tfidf_terms = ['surfactant','steric','hindering','cnt']

for i in range(len(new_df)):
    word_sum = 0
    for sentence in new_df[['content']].iloc[i]:
        split_sentence = sentence.split()
        for word in split_sentence:
```

```
            if word in tfidf_terms:
                word_sum += 1
    word_count.append(word_sum)

new_df['input'] = word_count
```

[332]:
```
tfidf_terms = ['marketability','reliability','cost','flux']
word_count = []
for i in range(len(new_df)):
    word_sum = 0
    for sentence in new_df[['content']].iloc[i]:
        split_sentence = sentence.split()
        for word in split_sentence:
            if word in tfidf_terms:
                word_sum += 1
    word_count.append(word_sum)

new_df['output'] = word_count
```

Now we group by user and then we can see how many times each student said a certain word.

[333]:
```
#Grouping by users
user_group = new_df.groupby(['userIDs'], as_index = False)
#Getting the sum of the contributions
user_sum = user_group.sum()
```

[334]:
```
user_sum['NewOutcomeScore'] = user_sum['OutcomeScore'] / user_group.
 ↪count()['OutcomeScore']
user_data = user_sum
user_data
```

[334]:

	userIDs	Unnamed: 0	Line_ID	group_id	m_experimental_testing	\
0	2	12286	12286	130	2	
1	3	2796	2796	44	1	
2	4	6454	6454	62	2	
3	5	9385	9385	90	0	
4	6	6740	6740	74	0	
..	...	...	...	...	...	
364	389	874511	874649	276	2	
365	390	913147	913291	288	0	
366	391	970395	970548	306	2	
367	392	684840	684948	216	2	
368	393	1369484	1369700	432	5	

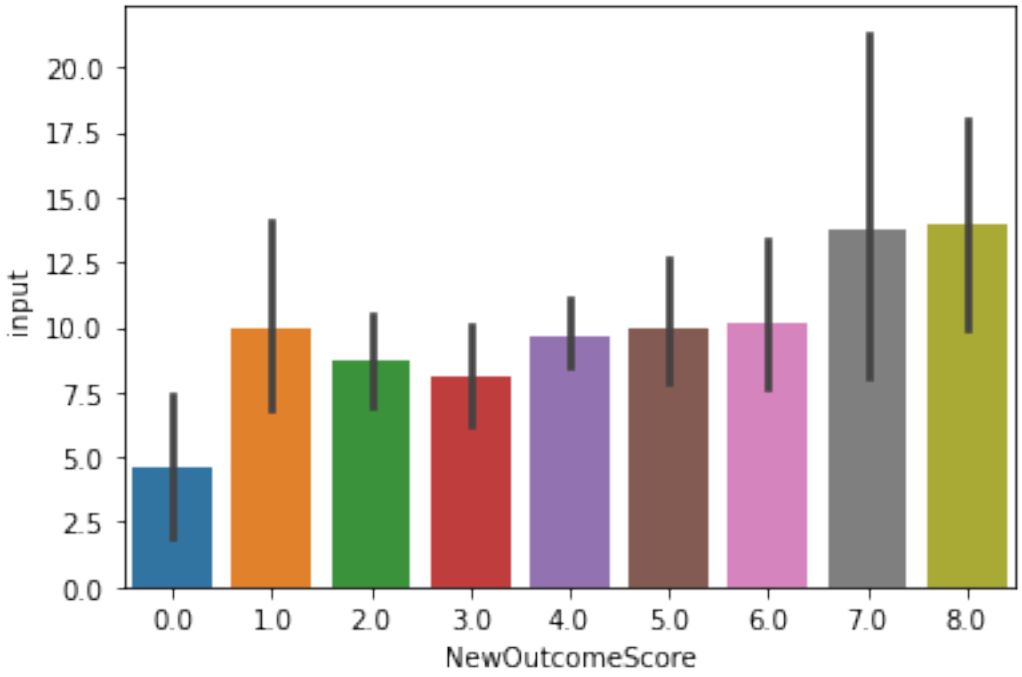
	m_making_design_choices	m_asking_questions	\
0	4	17	
1	4	3	
2	2	3	

```
3                          0                                  6
4                          2                                  7
..                       ...                                ...
364                        8                                  4
365                        5                                  6
366                        3                                 11
367                        4                                  4
368                        6                                  8

     j_customer_consultants_requests  j_performance_parameters_requirements  \
0                                  0                                       4
1                                  0                                       1
2                                  1                                       5
3                                  0                                       2
4                                  1                                       3
..                               ...                                     ...
364                                1                                       7
365                                1                                       2
366                                1                                       4
367                                0                                       2
368                                1                                       7

     j_communication  OutcomeScore  wordCount  input  output  NewOutcomeScore
0                  0           260        704      4      11              4.0
1                  0            88        157      2       9              4.0
2                  0           124        349      6      14              4.0
3                  0            90        342      3       7              2.0
4                  0            74        399      6      16              2.0
..               ...           ...        ...    ...     ...              ...
364                0           322        755     33       9              7.0
365                0           192        431      6       6              4.0
366                0           255        605     19      13              5.0
367                0           180        451      9       6              5.0
368                0           288        868     27      11              4.0

[369 rows x 15 columns]
```

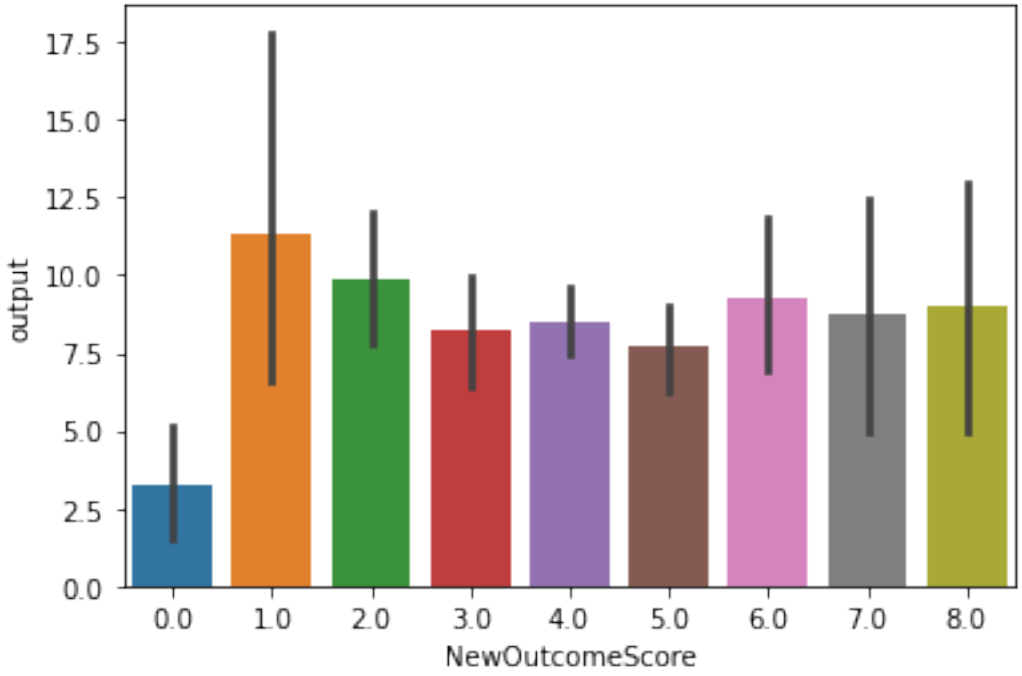
6.2.2 Plotting

We can plot the amount of times input and output terms were said on average per each outcome score and there seems to be a slight correlation with the higher marks.

[335]:
```
sns.barplot(data = user_data, x = 'NewOutcomeScore', y = "input");
```



[336]:
```
sns.barplot(data = user_data, x = 'NewOutcomeScore', y = "output");
```



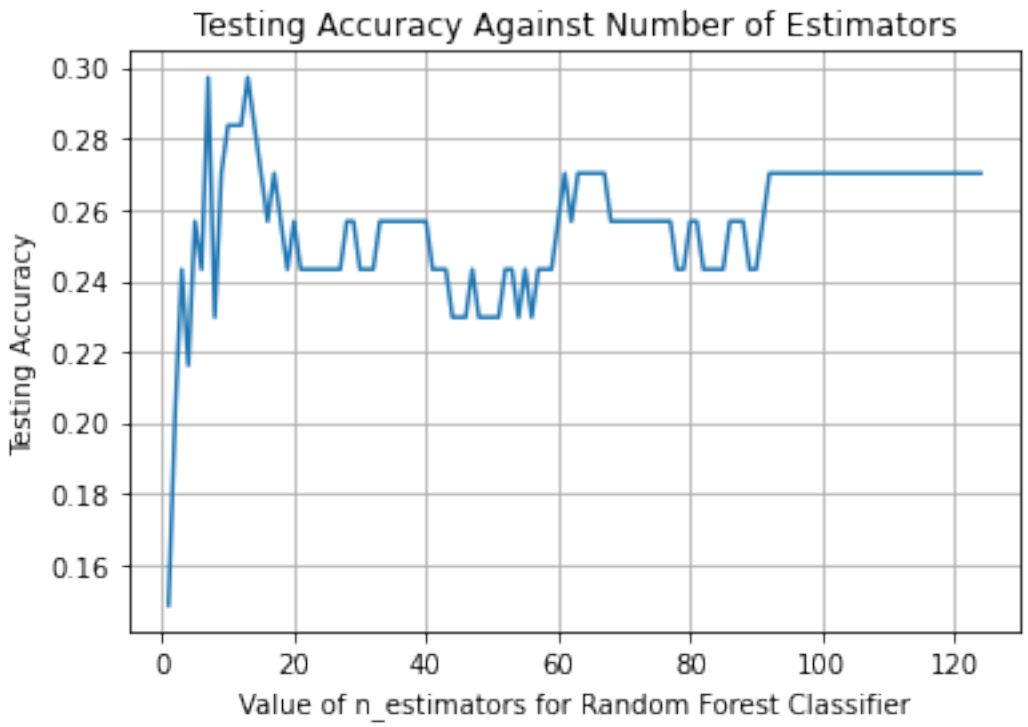
Using just new features

[337]:
```
X = user_data[['input','output']]
Y = user_data['NewOutcomeScore']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8,␣
 ↪random_state=42)

rfc = RandomForestClassifier(n_estimators = 20, max_samples = 0.5, random_state␣
 ↪= 42).fit(X_train, Y_train)
Y_pred = rfc.predict(X_test)
accuracy = np.round(accuracy_score(Y_test, Y_pred), 3)

print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.297
```

[338]:
```
scores =[]
for i in range(1, 125):
    rfc = RandomForestClassifier(n_estimators = i, random_state = 42)
    rfc.fit(X_train, Y_train)
    Y_pred = rfc.predict(X_test)
    scores.append(accuracy_score(Y_test, Y_pred))

plt.plot(range(1, 125), scores)
plt.xlabel('Value of n_estimators for Random Forest Classifier')
plt.ylabel('Testing Accuracy')
plt.title('Testing Accuracy Against Number of Estimators')
plt.grid(True); GridSearchCV
```

[338]:
```
sklearn.model_selection._search.GridSearchCV
```



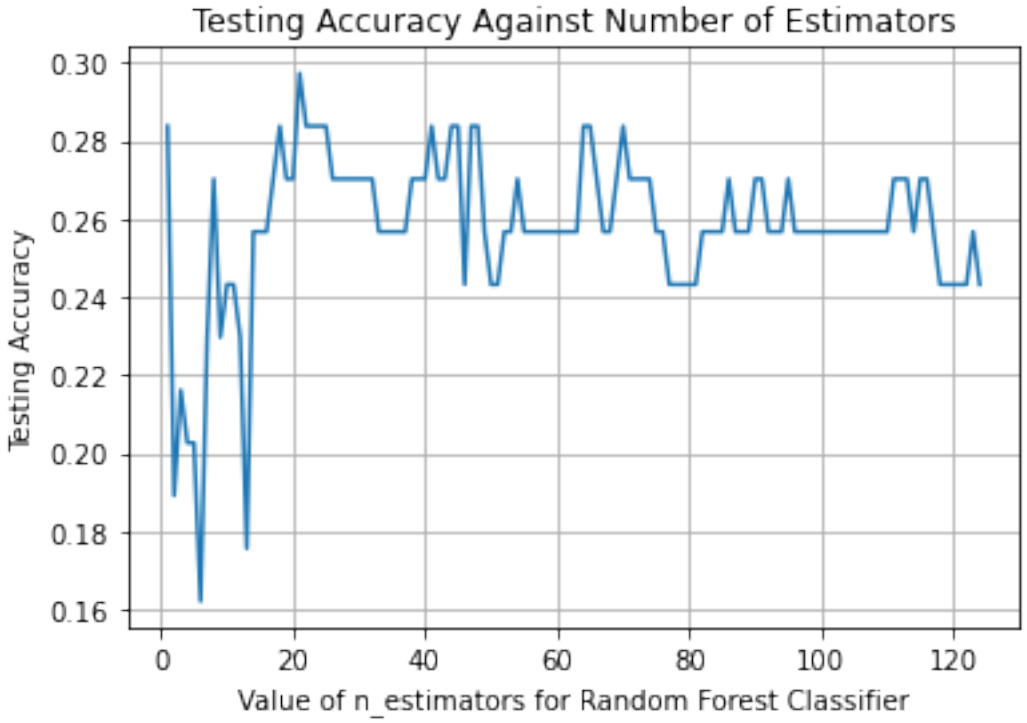
All Variables

[339]:
```
X = user_data.drop(['NewOutcomeScore', 'OutcomeScore','group_id','Line_ID',␣
 ↪'Unnamed: 0', 'userIDs'], axis = 1)
Y = user_data['NewOutcomeScore']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8,␣
 ↪random_state=42)

rfc = RandomForestClassifier(n_estimators = 24, max_samples = 0.5, random_state␣
 ↪= 42).fit(X_train, Y_train)
Y_pred = rfc.predict(X_test)
accuracy = np.round(accuracy_score(Y_test, Y_pred), 3)

print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.311
```

[340]:
```
scores =[]
for i in range(1, 125):
    rfc = RandomForestClassifier(n_estimators = i, random_state = 42)
    rfc.fit(X_train, Y_train)
    Y_pred = rfc.predict(X_test)
    scores.append(accuracy_score(Y_test, Y_pred))
```

```
plt.plot(range(1, 125), scores)
plt.xlabel('Value of n_estimators for Random Forest Classifier')
plt.ylabel('Testing Accuracy')
plt.title('Testing Accuracy Against Number of Estimators')
plt.grid(True);
```



Feature Importance

[341]:
```
for name, score in zip(X.columns, rfc.feature_importances_):
    print(name, np.round(score, 3))
```

```
m_experimental_testing 0.078
m_making_design_choices 0.117
m_asking_questions 0.138
j_customer_consultants_requests 0.062
j_performance_parameters_requirements 0.093
j_communication 0.046
wordCount 0.195
input 0.136
output 0.134
```

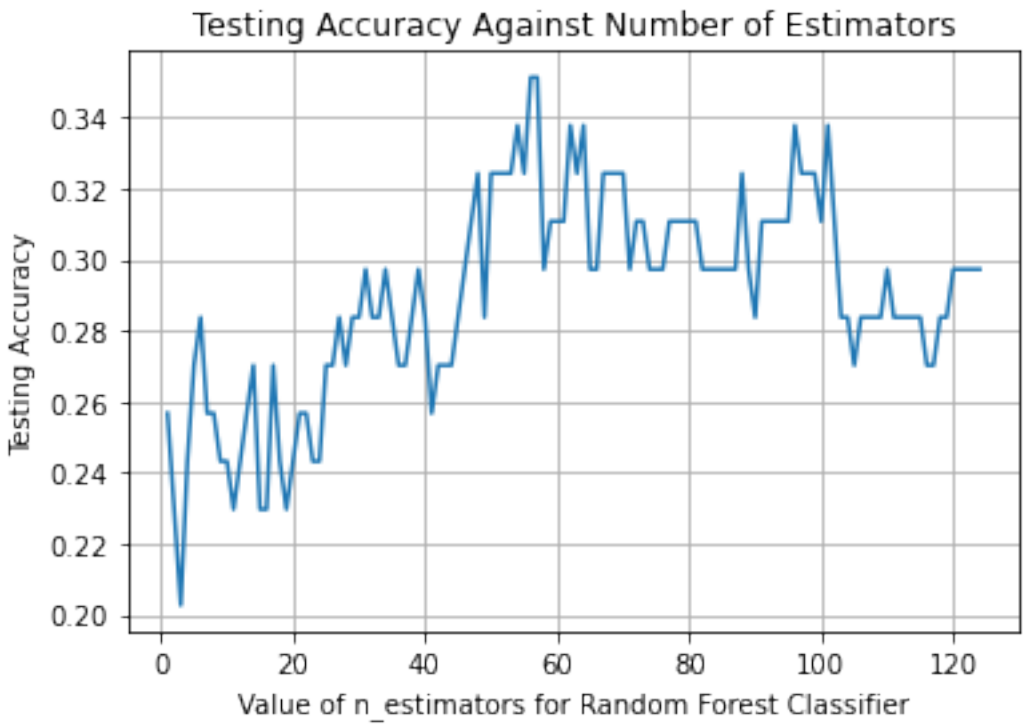
Five most important features.

[342]:
```
X = user_data[['wordCount', 'm_asking_questions', 'm_making_design_choices',␣
 ↪'input', 'output']]
Y = user_data['NewOutcomeScore']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8,␣
 ↪random_state=42)

rfc = RandomForestClassifier(n_estimators = 56, max_samples = 0.5, random_state␣
 ↪= 42).fit(X_train, Y_train)
Y_pred = rfc.predict(X_test)
accuracy = np.round(accuracy_score(Y_test, Y_pred), 3)

print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.297
```

[343]:
```
scores =[]
for i in range(1, 125):
    rfc = RandomForestClassifier(n_estimators = i, random_state = 42)
    rfc.fit(X_train, Y_train)
    Y_pred = rfc.predict(X_test)
    scores.append(accuracy_score(Y_test, Y_pred))

plt.plot(range(1, 125), scores)
plt.xlabel('Value of n_estimators for Random Forest Classifier')
plt.ylabel('Testing Accuracy')
plt.title('Testing Accuracy Against Number of Estimators')
plt.grid(True); GridSearchCV
```

[343]:
```
sklearn.model_selection._search.GridSearchCV
```


Testing Accuracy Against Number of Estimators
Testing Accuracy
0.34
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0
20
40
60
80
100
120
Value of n_estimators for Random Forest Classifier