Part 1: Language Modeling / Regression

```
import pandas as pd

path = r'C:\Users\ACER\PycharmProjects\LogisticRegression\env\NLP\
answers.csv'
answers_df = pd.read_csv(path)
```

```
        id                                                answer  score
correct
0      1.1  High risk problems are address in the prototyp...    3.5
0.0
1      1.1  To simulate portions of the desired final prod...    5.0
1.0
2      1.1  A prototype program simulates the behaviors of...    4.0
1.0
3      1.1  Defined in the Specification phase a prototype...    5.0
1.0
4      1.1  It is used to let the users have a first idea ...    3.0
0.0
...    ...                                                ...    ...
...
2437  12.1                                             log n    5.0
1.0
2438  12.1                                 minus 1 divided by 2    1.5
0.0
2439  12.1                                              2n-1    2.5
0.0
2440  12.1  it takes at most h steps, where h is the heigh...    5.0
1.0
2441  12.1  it depends on the install search tree then fro...    1.5
0.0

[2442 rows x 4 columns]
```

establishing NLP pipeline

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

import nltk

nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\ACER\AppData\Roaming\nltk_data...

True
```

```
import re


def preprocess_text(text):
    text = re.sub(r'[^\w\s]', '', text.lower())
    text = re.sub(r'\d', '', text)
    tokens = word_tokenize(text)
    filtered_tokens = [token for token in tokens if token not in
stopwords.words('english')]
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in
filtered_tokens]
    return lemmatized_tokens

answers_df['answer'] = [preprocess_text(text) for text in
answers_df['answer']]

answers_df.head(5)
```

```
    id                                                answer  score
correct
0  1.1  [high, risk, problem, address, prototype, prog...    3.5
0.0
1  1.1  [simulate, portion, desired, final, product, q...    5.0
1.0
2  1.1  [prototype, program, simulates, behavior, port...    4.0
1.0
3  1.1  [defined, specification, phase, prototype, sti...    5.0
1.0
4  1.1  [used, let, user, first, idea, completed, prog...    3.0
0.0
```

encoding the data using a pretrained model

the word2vec pre-trained Google News corpus (3 billion running words) word vector model (3 million 300-dimension English word vectors). here is the **link** to download the model.

```
from gensim.models import KeyedVectors

model_cbow = KeyedVectors.load_word2vec_format(
    r'C:\Users\ACER\PycharmProjects\LogisticRegression\env\NLP\
GoogleNews-vectors-negative300.bin', binary=True)

import numpy as np


def text_to_vector(text, model):
    word_vectors = []
    for word in text:
        if word in model.key_to_index:
```

```
            word_vectors.append(model[word])
    if len(word_vectors) > 0:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(model.vector_size)

answers_df_vectors = answers_df.copy()

answers_df_vectors['answer'] = [text_to_vector(text, model_cbow) for
text in answers_df_vectors['answer']]

answers_df_vectors.head(5)
```

```
    id                                             answer  score
correct
0  1.1  [-0.011313991, 0.009006701, 0.03616333, 0.0942...    3.5
0.0
1  1.1  [0.06038947, 0.0151013825, 0.0019430863, 0.078...    5.0
1.0
2  1.1  [-0.026572488, 0.0030524514, 0.023925781, 0.02...    4.0
1.0
3  1.1  [-0.008671352, 0.014805385, 0.00014241536, 0.0...    5.0
1.0
4  1.1  [-0.0045543853, -0.0039262315, 0.03354972, 0.0...    3.0
0.0
```

preparing the dataframe for train and testing

```
mat_answers_df = np.array([arr for arr in
answers_df_vectors['answer']])

answers_df_vectors.drop(['answer'], axis=1, inplace=True)

prepro_answers_df =
answers_df_vectors.join(pd.DataFrame(mat_answers_df))

prepro_answers_df.columns = prepro_answers_df.columns.astype(str)
prepro_answers_df.dropna(axis=1, inplace=True)

prepro_answers_df
```

```
        id  score  correct         0         1         2         3
4  \
0      1.1    3.5      0.0 -0.011314  0.009007  0.036163  0.094203 -
0.086490
1      1.1    5.0      1.0  0.060389  0.015101  0.001943  0.078427 -
0.051615
2      1.1    4.0      1.0 -0.026572  0.003052  0.023926  0.027011 -
0.124079
3      1.1    5.0      1.0 -0.008671  0.014805  0.000142  0.075033 -
0.145578
```

```
4      1.1    3.0      0.0 -0.004554 -0.003926  0.033550  0.072991 -
0.047009
...    ...    ...      ...       ...       ...       ...       ...
...
2437  12.1    5.0      1.0  0.059906 -0.001709 -0.172974  0.051758 -
0.112305
2438  12.1    1.5      0.0  0.057083 -0.097229  0.092163  0.070007 -
0.083618
2439  12.1    2.5      0.0  0.000000  0.000000  0.000000  0.000000
0.000000
2440  12.1    5.0      1.0 -0.039510  0.178182 -0.027710 -0.015666 -
0.085327
2441  12.1    1.5      0.0  0.078084  0.018366 -0.008784  0.052569 -
0.065615

             5         6  ...       290       291       292       293
\
0     0.018722  0.106346  ... -0.109144  0.056168 -0.094267 -0.002266

1    -0.006441  0.117803  ... -0.111039  0.037476 -0.037692  0.014641

2     0.011813  0.080247  ... -0.120128  0.113636 -0.021476 -0.043010

3     0.044373  0.101935  ... -0.168329  0.041958 -0.046039 -0.039089

4     0.024852  0.105057  ... -0.134584  0.128961 -0.055376  0.009431

...        ...       ...  ...       ...       ...       ...       ...

2437  0.082764  0.064453  ...  0.093750  0.042542  0.026489  0.014893

2438  0.013002 -0.010864  ... -0.066467  0.048859 -0.047043  0.087646

2439  0.000000  0.000000  ...  0.000000  0.000000  0.000000  0.000000

2440 -0.125814  0.041423  ... -0.049072  0.149902 -0.137634  0.075846

2441  0.034973  0.103816  ... -0.033778  0.135666 -0.024923  0.016905


           294       295       296       297       298       299
0    -0.000630  0.034838 -0.050432 -0.082519 -0.058517 -0.084094
1    -0.033205  0.092497 -0.027912 -0.017063 -0.050849 -0.066237
2     0.050251  0.023088  0.028648 -0.092174 -0.106146  0.061167
3    -0.018669  0.049331 -0.036801 -0.036714 -0.032854  0.002749
4    -0.027363  0.014968  0.045993 -0.082018 -0.065550 -0.084932
...        ...       ...       ...       ...       ...       ...
2437 -0.046875 -0.259277  0.060059 -0.158569 -0.154785 -0.022278
2438 -0.072632 -0.008606 -0.140869 -0.116821 -0.064392  0.026764
2439  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
```

```
2440 -0.028071 -0.089600 -0.154867 -0.002360 -0.088175  0.027323
2441  0.022451 -0.023905 -0.034216 -0.109090  0.036997  0.015066

[2442 rows x 303 columns]
```

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test =
train_test_split(prepro_answers_df.drop(['score'], axis=1),

prepro_answers_df['score'], test_size=0.2, random_state=20)
```

training and hyperparameter tuning using GridSearch

```
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error,
root_mean_squared_error

# Define a dictionary to store model names and their respective
hyperparameter grids
model_params = {
    'SVR': {
        'kernel': ['rbf', 'linear'],
        'C': [0.1, 1, 10],
        'gamma': [0.01, 0.1, 1]
    },
    'Linear Regression': {},
    'Decision Tree': {
        'max_depth': [2, 5, 10],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
}

models = []
model_names = []
mse_scores = []
mae_scores = []
rmse_scores = []

for model_name, params in model_params.items():
    # Create the model object
    if model_name == 'SVR':
        model = SVR()
    elif model_name == 'Linear Regression':
        model = LinearRegression()
    else:
```

```
        model = DecisionTreeRegressor()

    print(model_name)
    grid_search = GridSearchCV(model, params,
scoring='neg_mean_squared_error', cv=5, verbose=10)

    grid_search.fit(x_train, y_train)
    best_model = grid_search.best_estimator_
    preds = best_model.predict(x_test)

    mse = mean_squared_error(y_test, preds)
    mae = mean_absolute_error(y_test, preds)
    rmse = root_mean_squared_error(y_test, preds)

    models.append(best_model)
    model_names.append(model_name)
    mse_scores.append(mse)
    mae_scores.append(mae)
    rmse_scores.append(rmse)
```

result of regression

```
print("Model Name\tMSE\t\tMAE\t\tRMSE")
print("-----------------------------------------")
for i in range(len(models)):
    print(f"{model_names[i]:<12}\t{mse_scores[i]:.4f}\
t{mae_scores[i]:.4f}\t{rmse_scores[i]:.4f}")
```

```
Model Name MSE        MAE        RMSE
-----------------------------------------
SVR             0.2108     0.3050     0.4592
Linear Regression     0.2673     0.3788     0.5170
Decision Tree   0.2630     0.3723     0.5129
```

```
models[0]
```

```
SVR(C=10, gamma=0.1)
```

looking at the results, it that SVM is the best model suited for the task

Part 2: Language Modeling / Classification

```
path = r'C:\Users\ACER\PycharmProjects\LogisticRegression\env\NLP'
tweets_test_df = pd.read_csv(path + r'\twitter_validation.csv',
                             names=['tweet_id', 'entity', 'sentiment',
'Tweet content'])

tweets_test_df.dropna(inplace=True)

tweets_test_df['Tweet content'] = [preprocess_text(text) for text in
tweets_test_df['Tweet content']]
```

```
tweets_test_df_vectors = tweets_test_df.copy()
print(tweets_test_df_vectors.head(5))

tweets_test_df_vectors['Tweet content'] = [text_to_vector(text,
model_cbow) for text in

tweets_test_df_vectors['Tweet content']]

print(tweets_test_df_vectors.head(5))

mat_validation_df = np.array([arr for arr in
tweets_test_df_vectors['Tweet content']])
print(mat_validation_df[:2])

prepro_validation_df =
tweets_test_df_vectors.join(pd.DataFrame(mat_validation_df))
prepro_validation_df.drop(['Tweet content'], axis=1, inplace=True)
print(prepro_validation_df.head(5))

prepro_validation_df.columns =
prepro_validation_df.columns.astype(str)

prepro_validation_df.dropna(axis=0, inplace=True)

print(prepro_validation_df.columns)

Index(['tweet_id', 'entity', 'sentiment', '0', '1', '2', '3', '4',
'5', '6',
       ...
       '290', '291', '292', '293', '294', '295', '296', '297', '298',
'299'],
      dtype='object', length=303)

prepro_validation_df.drop(['tweet_id', 'entity'], axis=1,
inplace=True)

print(prepro_validation_df.head(5))

from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
prepro_validation_df['sentiment'] =
encoder.fit_transform(prepro_validation_df['sentiment'])

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test =
train_test_split(prepro_validation_df.drop(['sentiment'], axis=1),

prepro_validation_df['sentiment'], test_size=0.2, random_state=20)
```

```
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report

model_params = {
    'SVM': {
        'kernel': ['linear', 'rbf', 'poly'],
        'C': [0.1, 1, 10]
    },
    'D.T': {
        'criterion': ["gini", "log_loss", "entropy"],
        'max_depth': [5, 10, 15]
    },
    'Logistic Regression': {
        'solver': ['lbfgs', 'liblinear']
    },
    'AdaBoost': {
        'n_estimators': [50, 100, 200]
    }
}

models = []
model_names = []
reports = []

for model_name, params in model_params.items():

    if model_name == 'SVM':
        model = SVC(probability=True,random_state=10)
    elif model_name == 'D.T':
        model = DecisionTreeClassifier(random_state=10)
    elif model_name == 'Logistic Regression':
        model = LogisticRegression(random_state=10)
    else:
        model = AdaBoostClassifier( algorithm="SAMME",random_state=10)

    grid_search = GridSearchCV(model, params, cv=5, verbose=10)
    grid_search.fit(x_train, y_train)
    best_model = grid_search.best_estimator_
    preds = best_model.predict(x_test)
    report = classification_report(y_test, preds)
    models.append(best_model)
    model_names.append(model_name)
    reports.append(report)


for i in range(len(models)):
```

```
    print(f"{model_names[i]} Report:\n")
    print(reports[i])
    print("--------------------------------------------")
```

```
SVM Report:

              precision    recall  f1-score   support

           0       0.43      0.36      0.39        33
           1       0.46      0.53      0.49        57
           2       0.43      0.39      0.41        54
           3       0.50      0.52      0.51        56

    accuracy                           0.46       200
   macro avg       0.45      0.45      0.45       200
weighted avg       0.46      0.46      0.46       200

--------------------------------------------
D.T Report:

              precision    recall  f1-score   support

           0       0.27      0.21      0.24        33
           1       0.50      0.33      0.40        57
           2       0.43      0.69      0.53        54
           3       0.44      0.39      0.42        56

    accuracy                           0.42       200
   macro avg       0.41      0.41      0.40       200
weighted avg       0.43      0.42      0.41       200

--------------------------------------------
Logistic Regression Report:

              precision    recall  f1-score   support

           0       0.59      0.30      0.40        33
           1       0.51      0.53      0.52        57
           2       0.47      0.46      0.47        54
           3       0.49      0.62      0.55        56

    accuracy                           0.50       200
   macro avg       0.52      0.48      0.48       200
weighted avg       0.51      0.50      0.49       200

--------------------------------------------
AdaBoost Report:

              precision    recall  f1-score   support

           0       0.41      0.27      0.33        33
```

```
           1       0.53      0.40      0.46        57
           2       0.35      0.48      0.41        54
           3       0.49      0.54      0.51        56

    accuracy                           0.44       200
   macro avg       0.45      0.42      0.43       200
weighted avg       0.45      0.44      0.44       200

--------------------------------------------
```

While the **Logistic Regression** model achieved the highest accuracy of 0.50, all models tested resulted in a relatively low accuracy range between 0.42 and 0.50. This suggests that further exploration and potentially different modeling techniques might be necessary to achieve better performance for this specific task.