

Final Prediction Contest (woo!)

Q1) I used a conventional neural network (implemented in tensor flow) with three hidden layers in order to solve this problem. For the word embeddings I used MiniLM to predict the embeddings for the input data. I trained the model on data adversarial generated using FGSM, and just with Guassian noise as well!

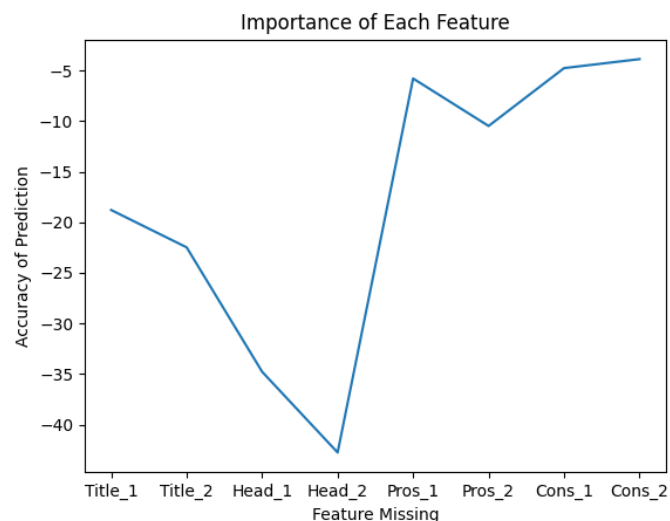
My model had the following performance on the training set:

$$Accuracy = 83.18\%$$

This produced the following Confusion Matrix on the training data:

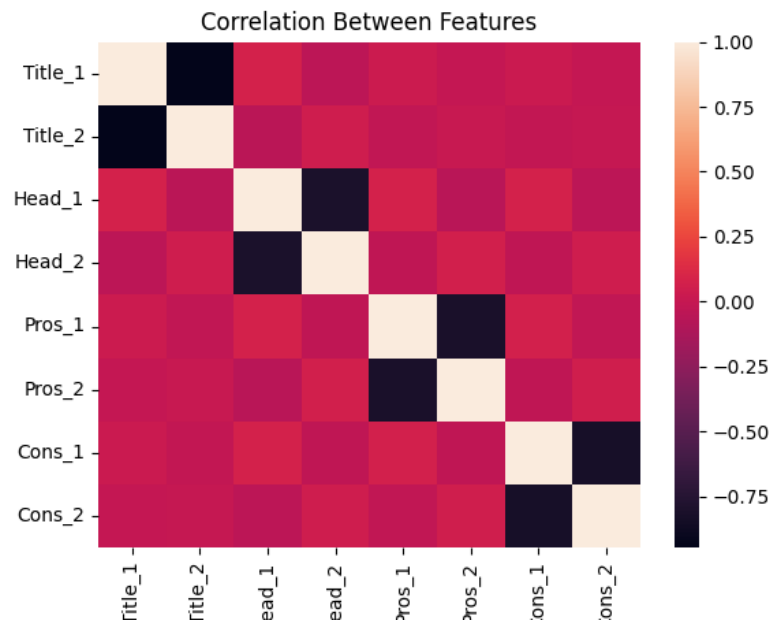
		Predicted Value					Total
		1	2	3	4	5	
Actual Value	1	34428	8315	242	4	2	42991
	2	1764	38749	15526	116	2	56157
	3	9	1208	111838	22883	69	136007
	4	0	1	6943	166408	11114	184466
	5	0	0	35	24425	109147	133607
Total		36201	48723	134584	213836	120334	553678

Q2) For the first figure, we want to show how "important" each figure is in a prediction. While the adversarial trained model trains on data that is missing predictions (as to mimic missing data entries on the test set) for the sake of this graph we are going to use the base model on the training set. That produces this graph, where the y axis represents difference in test accuracy with the embedding missing:



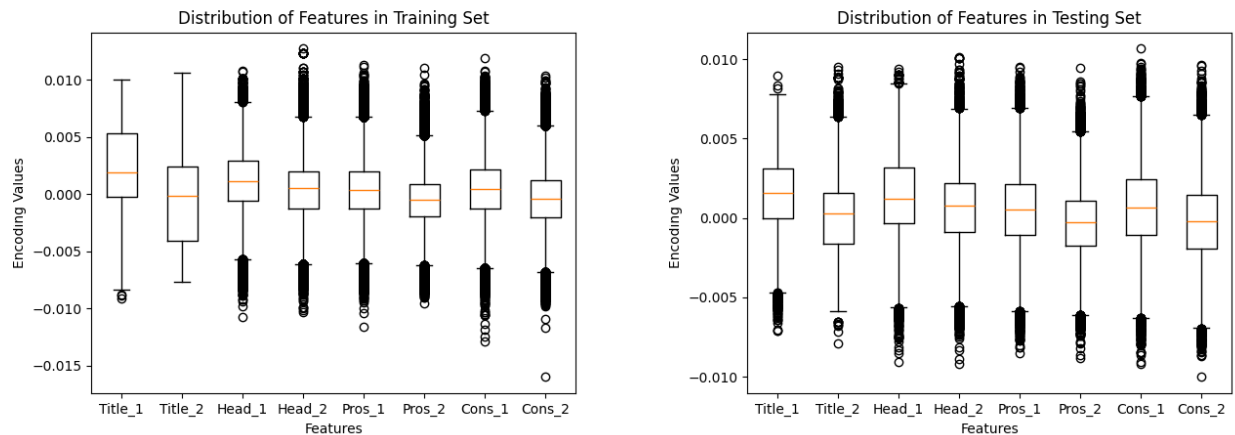
The graph above shows us that accuracy is semi dependant on every feature, but really maps to the heading the most. The accuracy is still pretty high (40%) even with the most important feature missing.

For the correlation between the features I had to use a mean pool for the embeddings as otherwise we would have way too many features to parse. This produced the following graph:



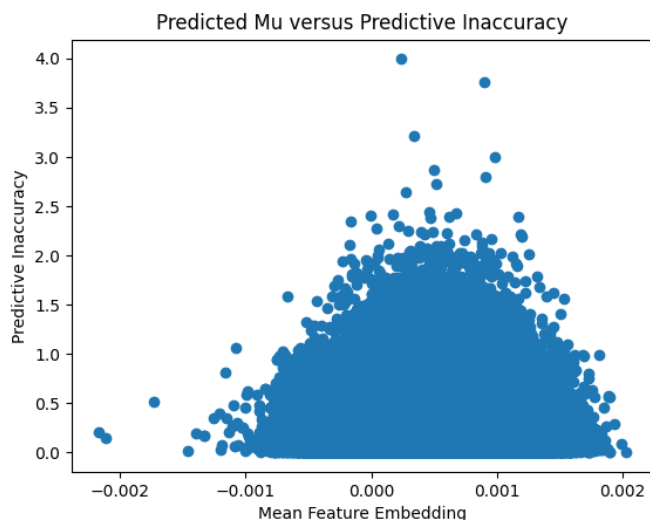
From this we can see that each of the double encodings are strongly correlated with each other, and that title tends to be more strongly correlated with heading then with pros.

I used a mean pool on the feature set for both test and training. This gives us the following distribution of features:

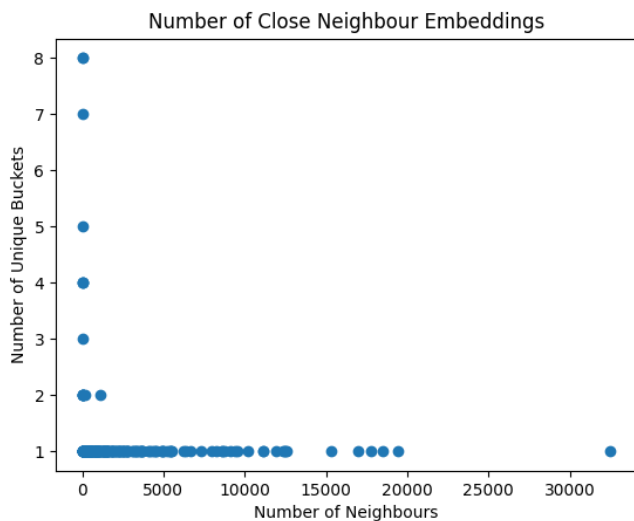


We can see that very roughly the means tend to stay the same across the training and test set. This means that with enough adversarial training we should be able to predict these features pretty robustly!

To view how common errors are, I decided to look at the mean value of the embedding (across all features) and the y value that was predicted. The y axis shows the difference between the predicted and the expected:



What this shows us is that embedding that tend to be more positive also tend to be predicted incorrectly. This also shows us that our algorithm tends to be off by 1 rather than off by a much larger amount. For the next graph, I wanted to compare how many nodes have an l2 norm of 0.02 from each other. However comparing this directly would have taken 2 days, so instead I will approximate this. To do this I will take our embedding space and break it into thresholds of 0.02, I will then count within how many values are within each of these squares to get an estimate for clustering.

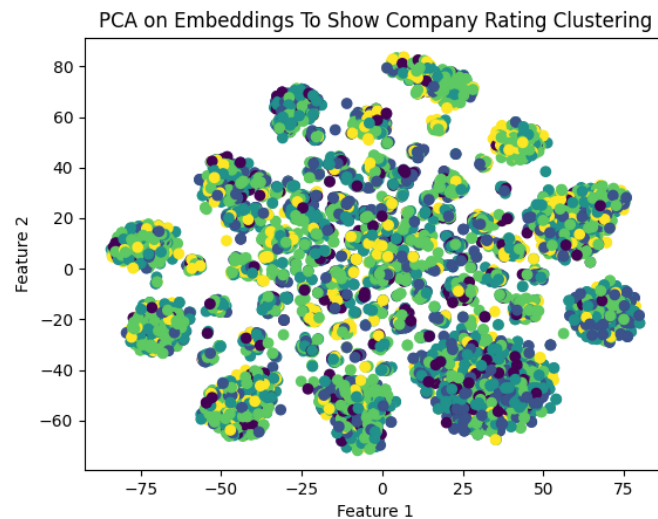


This demonstrates how similar certain embeddings are, and by proxy certain language and sentences should be as well. A large clustering of neighbours should correspond to a similar phrase or quandary being brought up (useful for 3!)

Q3) Firms (very broadly) can do one of two things in order to improve:

1. Avoid doing things that employees hate (Focus on cons for low rated companies).
2. Try to do more things that employees like (Focus on pros for high rated companies).

To do this we will be doing kmeans clustering on a sample of high rated companies and doing kmeans clustering on a sample of low rated companies. Below after doing PCA shows the rough distribution of company scores:



After this we will do kmeans clustering on the sample data (since the dimensions are so large this is not something we can visualize easily). However once we find the mean of some of our clusters we can use our dictionary to convert some of the encodings back into sentences. Some examples from the negative clustering:

1. ... Being at the bottom of the totem pole (in an entry level role) I do not get much, if any, **input on issues**...
2. There are many layers of management, so be prepared to be the **lowest person on the totem pole for quite some time**. Pay is good, but probably will not attract graduates from top business schools. Benefits are average and there are really no perks to be had.

And for the positive it includes:

1. ... that is basically a locker room, a shower, a field and some outdoor **basketball courts**...
2. ... benefits package - proclub, health insurance, dental coverage. large campus with football, **basketball**, volleyball fields - lot of sports leagues, flexible time, option to work remotely, presence all over the world...
3. Stability, benefits, brilliant coworkers, a sense of pride, Seattle (a wonderful town), a competitive Type-A atmosphere, a heck of a **soccer field**, good (if not free) food.

So two takeaways based on these clusters is that having more sports courts and facilities can help improve the ranking of a company. More over allowing employees to have more input also can allow the company to be rated higher.

Q4) To solve Q4 ChatGPT was given the prompt:

Given a series of language embedding, how can I identify small clusters corresponding to certain response variables?

ChatGPT provided the following response:

You're working with language embeddings and you want to form clusters based on response variables, I assume you're dealing with some sort of unsupervised learning problem, such as clustering.

Here's a general approach to perform clustering:

Data Preparation:

Ensure that you have a series of language embeddings for your data. Each data point should be represented by a vector in an embedding space.

Choose a Clustering Algorithm:

Select a clustering algorithm based on your specific requirements.

Common choices include K-means, hierarchical clustering, DBSCAN, or more advanced techniques like spectral clustering or Gaussian mixture models.

Determine the Number of Clusters (K):

If you're using algorithms like K-means, you'll need to specify the number of clusters, K. You can use techniques like the elbow method or silhouette analysis to find an optimal value for K.

Apply the Clustering Algorithm:

Use your chosen algorithm to cluster the data points based on their language embeddings. Each cluster will represent a group of similar data points.

Analysis and Interpretation:

Once the clustering is done, analyze the clusters to understand their characteristics. You can look at the response variables associated with each cluster to see if there are patterns or trends.

Validation and Refinement:

Evaluate the quality of your clusters. You might need to refine the clustering parameters or try different algorithms to achieve better results.

Here's a Python example using the popular K-means clustering algorithm with scikit-learn

```
python
Copy code
from sklearn.cluster import KMeans
import numpy as np

# Assuming X contains your language embeddings
```

```
X = np.array([[...], [...], ...])

# Choose the number of clusters (K)
k = 3

# Apply K-means clustering
kmeans = KMeans(n_clusters=k, random_state=42)
labels = kmeans.fit_predict(X)

# Now 'labels' contains the cluster assignments for each data point
Remember to adapt the code to your specific needs and dataset. Also, consider
normalizing or scaling your data before applying clustering algorithms,
depending on the characteristics of your embeddings.
```

Code:

```
import pickle
import pandas as pd
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from tensorflow.keras import layers, models
from sklearn.metrics import r2_score, mean_squared_error, accuracy_score
from sklearn.manifold import TSNE

# ===== Load Embeddings to use =====

embeddings = {}

with open('embeddings.pkl', "rb") as fIn:
    stored_data = pickle.load(fIn)

    stored_sentences = stored_data['sentences']
    stored_embeddings = stored_data['embeddings']

    for i in tqdm(range(len(stored_sentences))):
        embeddings[stored_sentences[i]] = stored_embeddings[i]

with open('testEmbeddings.pkl', "rb") as fIn:
    stored_data = pickle.load(fIn)

    stored_sentences = stored_data['sentences']
    stored_embeddings = stored_data['embeddings']

    for i in tqdm(range(len(stored_sentences))):
        embeddings[stored_sentences[i]] = stored_embeddings[i]

# We need to take inputs in for job_title, headline, pros, cons
embedding_size = len(embeddings["Software Engineer"])
input_layer_size = embedding_size * 5

# ===== Create the Model =====

model = models.Sequential()

model.add(layers.Dense(1024, activation='relu', input_shape=(input_layer_size,)))
model.add(layers.Dropout(.1))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1))

model.compile(
    optimizer=tf.keras.optimizers.Adam(),
    loss=tf.keras.losses.MeanSquaredError(),
    metrics=[tf.keras.metrics.MeanSquaredError()],
)

# ===== Load the Data =====
```

```

def cleanData(df):
    X_Train = []
    Y_Train = df["overall_rating"].to_numpy()

    for _, row in tqdm(df.iterrows()):
        items = [[] for item in range(5)]

        if row["job_title"] in embeddings:
            items[1] = embeddings[row["job_title"]]
        else:
            items[1] = [0] * embedding_size

        if row["headline"] in embeddings:
            items[2] = embeddings[row["headline"]]
        else:
            items[2] = [0] * embedding_size

        if row["pros"] in embeddings:
            items[3] = embeddings[row["pros"]]
        else:
            items[3] = [0] * embedding_size

        if row["cons"] in embeddings:
            items[4] = embeddings[row["cons"]]
        else:
            items[4] = [0] * embedding_size

        X_Train.append(np.array(items).flatten())

    X_Train = np.array(X_Train)

    return X_Train, Y_Train

# 5% of the time return nothing for a given entry
TOL = 5

def generateSkewedX(df):
    X_Train = []
    Y_Train = df["overall_rating"].to_numpy()

    for _, row in tqdm(df.iterrows()):
        items = [[] for item in range(5)]

        if np.random.randint(0, 100) > TOL and row["job_title"] in embeddings:
            items[1] = embeddings[row["job_title"]]
        else:
            items[1] = [0] * embedding_size

        if np.random.randint(0, 100) > TOL and row["headline"] in embeddings:
            items[2] = embeddings[row["headline"]]
        else:
            items[2] = [0] * embedding_size

        if np.random.randint(0, 100) > TOL and row["pros"] in embeddings:
            items[3] = embeddings[row["pros"]]
        else:
            items[3] = [0] * embedding_size

```



```

if np.random.randint(0, 100) > TOL and row["cons"] in embeddings:
    items[4] = embeddings[row["cons"]]
else:
    items[4] = [0] * embedding_size

for x in range(0, embedding_size):
    items[0][x] += np.random.normal(0, 0.05)
    items[1][x] += np.random.normal(0, 0.05)
    items[2][x] += np.random.normal(0, 0.05)
    items[3][x] += np.random.normal(0, 0.05)
    items[4][x] += np.random.normal(0, 0.05)
X_Train.append(np.array(items).flatten())

X_Train = np.array(X_Train)

return X_Train, Y_Train

# ===== Run the Tests =====
trainingSet = pd.read_csv("Econ424_F2023_PC6_glassdoor_training_large_v1.csv")
testingSet = pd.read_csv("424_F2023_Final_PC_glassdoor_test_without_response_v1.csv")
testingResponse = pd.read_csv("PC6_response.csv")

adv_model = tf.keras.models.load_model('trainedModels/v5-1.keras')

x_train, _ = cleanData(testingSet)

prediction_adv_train = np.round(adv_model.predict(x_train))

for loop in range(0, 10):
    for rep in range(0, 5):
        adv_xtrain = x_train + np.random.normal(0, 0.08, size=x_train.shape)
        adv_model.fit(adv_xtrain, y_train, epochs=2)

    adv_model.fit(x_train, y_train, epochs=2)
    adv_model.save("trainedModels/v5-" + str(loop) + ".keras")

x_test, _ = cleanData(testingSet)
y_test = testingResponse["overall_rating"].to_numpy()

norm_model = tf.keras.models.load_model('trainedModels/modelAt3.h5')

prediction_adv_train = adv_model.predict(x_train)
prediction_norm_train = norm_model.predict(x_train)

prediction_adv_test = adv_model.predict(x_test)
prediction_norm_test = norm_model.predict(x_test)

# Round the data using thresholds

np.savetxt("threshold.csv", prediction_adv_test, delimiter=",")

```

```

print("Train Data (R^2)")
print("Norm Model: " + str(r2_score(y_train, prediction_norm_train)))
print("Adv Model: " + str(r2_score(y_train, prediction_adv_train)))

print("Train Data (MSE)")
print("Norm Model: " + str(mean_squared_error(y_train, prediction_norm_train)))
print("Adv Model: " + str(mean_squared_error(y_train, prediction_adv_train)))

print("Test Data (R^2)")
print("Norm Model: " + str(r2_score(y_test, prediction_norm_test)))
print("Adv Model: " + str(r2_score(y_test, prediction_adv_test)))

print("Test Data (MSE)")
print("Norm Model: " + str(mean_squared_error(y_test, prediction_norm_test)))
print("Adv Model: " + str(mean_squared_error(y_test, prediction_adv_test)))

prediction = np.zeros((5, 5))

testPredictions = threshold(adv_testing_data)
# ===== To Export =====
f = open('ECON424_FINAL_PREDICTIONS.csv', 'w')

for estimate in testPredictions:
    f.writelines(str(estimate) + ",\n")

# ===== Graphs =====

trainingSet = pd.read_csv("Econ424_F2023_PC6_glassdoor_training_large_v1.csv")
testingSet = pd.read_csv("424_F2023_Final_PC_glassdoor_test_without_response_v1.csv")
adv_model = tf.keras.models.load_model('trainedModels/v5-1.keras')

x_train, y_train = cleanData(trainingSet)
x_test, _ = cleanData(testingSet)

no_title1 = np.copy(x_train)
no_title1[:, embedding_size: int(embedding_size * 3 / 2)] = 0
no_title2 = np.copy(x_train)
no_title2[:, int(embedding_size * 3 / 2): embedding_size * 2] = 0

no_headline1 = np.copy(x_train)
no_headline1[:, embedding_size * 2: int(embedding_size * 5 / 2)] = 0
no_headline2 = np.copy(x_train)
no_headline2[:, int(embedding_size * 5 / 2): embedding_size * 3] = 0

no_pros1 = np.copy(x_train)
no_pros1[:, embedding_size * 3: int(embedding_size * 7 / 2)] = 0
no_pros2 = np.copy(x_train)
no_pros2[:, int(embedding_size * 7 / 2): embedding_size * 4] = 0

no_cons1 = np.copy(x_train)
no_cons1[:, embedding_size * 4: int(embedding_size * 9 / 2)] = 0
no_cons2 = np.copy(x_train)

no_cons2[:, int(embedding_size * 9 / 2): embedding_size * 5] = 0

base = accuracy_score(y_train, adv_model.predict(x_train))

```

```

print(base)

predict_1 = accuracy_score(y_train, (adv_model.predict(no_title1)))
predict_2 = accuracy_score(y_train, (adv_model.predict(no_title2)))
predict_3 = accuracy_score(y_train, (adv_model.predict(no_headline1)))
predict_4 = accuracy_score(y_train, (adv_model.predict(no_headline2)))
predict_5 = accuracy_score(y_train, (adv_model.predict(no_pros1)))
predict_6 = accuracy_score(y_train, (adv_model.predict(no_pros2)))
predict_7 = accuracy_score(y_train, (adv_model.predict(no_cons1)))
predict_8 = accuracy_score(y_train, (adv_model.predict(no_cons2)))

xvalues = [1, 2, 3, 4, 5, 6, 7, 8]
yvalues = [predict_1 - base, predict_2 - base, predict_3 - base, predict_4 - base, predict_5 -
           base,
           predict_6 - base, predict_7 - base, predict_8 - base]

my_xticks = ['Title_1', 'Title_2', 'Head_1', 'Head_2', 'Pros_1', 'Pros_2', 'Cons_1', 'Cons_2']
plt.xticks(xvalues, my_xticks)
plt.ylabel("Accuracy of Prediction")
plt.xlabel("Feature Missing")
plt.plot(xvalues, yvalues)
plt.title("Importance of Each Feature")
plt.show()

f1 = []
f2 = []
f3 = []
f4 = []
f5 = []
f6 = []
f7 = []
f8 = []

for row in range(0, len(x_train)):
    f1.append(np.mean(x_train[row][embedding_size : int(embedding_size * 3 / 2)]))
    f2.append(np.mean(x_train[row][int(embedding_size * 3 / 2): embedding_size*2]))

    f3.append(np.mean(x_train[row][embedding_size*2: int(embedding_size * 5 / 2)]))
    f4.append(np.mean(x_train[row][int(embedding_size * 5 / 2): embedding_size * 3]))

    f5.append(np.mean(x_train[row][embedding_size*3: int(embedding_size * 7 / 2)]))
    f6.append(np.mean(x_train[row][int(embedding_size * 7 / 2): embedding_size * 4]))

    f7.append(np.mean(x_train[row][embedding_size*4: int(embedding_size * 9 / 2)]))
    f8.append(np.mean(x_train[row][int(embedding_size * 9 / 2): embedding_size * 5]))

corData = pd.DataFrame({'Title_1': f1, 'Title_2': f2, 'Head_1': f3, 'Head_2': f4, 'Pros_1': f5,
                        'Pros_2': f6, 'Cons_1': f7, 'Cons_2': f8})

sns.heatmap(corData.corr())
plt.title('Correlation Between Features')
plt.show()

plt.boxplot(corData.values, labels=corData.columns)
plt.title("Distribution of Features in Training Set")
plt.xlabel("Features")
plt.ylabel("Encoding Values")
plt.show()

```

```

f1 = []
f2 = []
f3 = []
f4 = []
f5 = []
f6 = []
f7 = []
f8 = []

for row in range(0, len(x_test)):
    f1.append(np.mean(x_test[row][embedding_size : int(embedding_size * 3 / 2)]))
    f2.append(np.mean(x_test[row][int(embedding_size * 3 / 2): embedding_size*2]))

    f3.append(np.mean(x_test[row][embedding_size*2: int(embedding_size * 5 / 2)]))
    f4.append(np.mean(x_test[row][int(embedding_size * 5 / 2): embedding_size * 3]))

    f5.append(np.mean(x_test[row][embedding_size*3: int(embedding_size * 7 / 2)]))
    f6.append(np.mean(x_test[row][int(embedding_size * 7 / 2): embedding_size * 4]))

    f7.append(np.mean(x_test[row][embedding_size*4: int(embedding_size * 9 / 2)]))
    f8.append(np.mean(x_test[row][int(embedding_size * 9 / 2): embedding_size * 5]))

corData = pd.DataFrame({'Title_1': f1, 'Title_2': f2, 'Head_1': f3, 'Head_2': f4, 'Pros_1': f5,
                        'Pros_2': f6, 'Cons_1': f7, 'Cons_2': f8})

plt.boxplot(corData.values, labels=corData.columns)
plt.title("Distribution of Features in Testing Set")
plt.xlabel("Features")
plt.ylabel("Encoding Values")
plt.show()
prediction = adv_model.predict(x_train)
x_axis = []
y_axis = []

for row in range(0, len(x_train)):
    x_axis.append(np.mean(x_train[row]))
    y_axis.append(np.abs(y_train[row] - prediction[row]))

plt.scatter(x_axis, y_axis)
plt.xlabel("Mean Feature Embedding")
plt.ylabel("Predictive Inaccuracy")
plt.title("Predicted Mu versus Predictive Inaccuracy")
plt.show()
data = {}

scores = np.zeros((4,4,4,4,4,4,4,4))
f1 = []
f2 = []
f3 = []
f4 = []
f5 = []
f6 = []
f7 = []
f8 = []

```

```

def retVal(val):
    if (val < -0.02):
        return 0
    elif val < 0:
        return 1
    elif val < 0.02:
        return 2
    else:
        return 4

for row in tqdm(range(0, len(x_train))):

    f1 = retVal(np.mean(x_train[row][embedding_size : int(embedding_size * 3 / 2)]))
    f2 = retVal(np.mean(x_train[row][int(embedding_size * 3 / 2): embedding_size*2]))

    f3 = retVal(np.mean(x_train[row][embedding_size*2: int(embedding_size * 5 / 2)]))
    f4 = retVal(np.mean(x_train[row][int(embedding_size * 5 / 2): embedding_size * 3]))

    f5 = retVal(np.mean(x_train[row][embedding_size*3: int(embedding_size * 7 / 2)]))
    f6 = retVal(np.mean(x_train[row][int(embedding_size * 7 / 2): embedding_size * 4]))

    f7 = retVal(np.mean(x_train[row][embedding_size*4: int(embedding_size * 9 / 2)]))
    f8 = retVal(np.mean(x_train[row][int(embedding_size * 9 / 2): embedding_size * 5]))

    scores[f1][f2][f3][f4][f5][f6][f7][f8] += 1

scores = scores.flatten()

data = {}

for score in scores:
    if score != 0:
        if score not in data:
            data[score] = 1
        else:
            data[score] += 1

plt.scatter(list(data.keys()), list(data.values()))
plt.xlabel("Number of Neighbours")
plt.ylabel("Number of Unique Buckets")
plt.title("Number of Close Neighbour Embeddings")
plt.show()

x_train = x_train[0:10000]
# Perform t-SNE dimensionality reduction to 2D
tsne = TSNE(n_components=2, n_jobs=10, verbose=1)
embeddings_2d = tsne.fit_transform(x_train)

plt.scatter(embeddings_2d[:,0], embeddings_2d[:,1], c=y_train[0:10000], cmap='viridis')
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("PCA on Embeddings To Show Company Rating Clustering")
plt.show()

```
