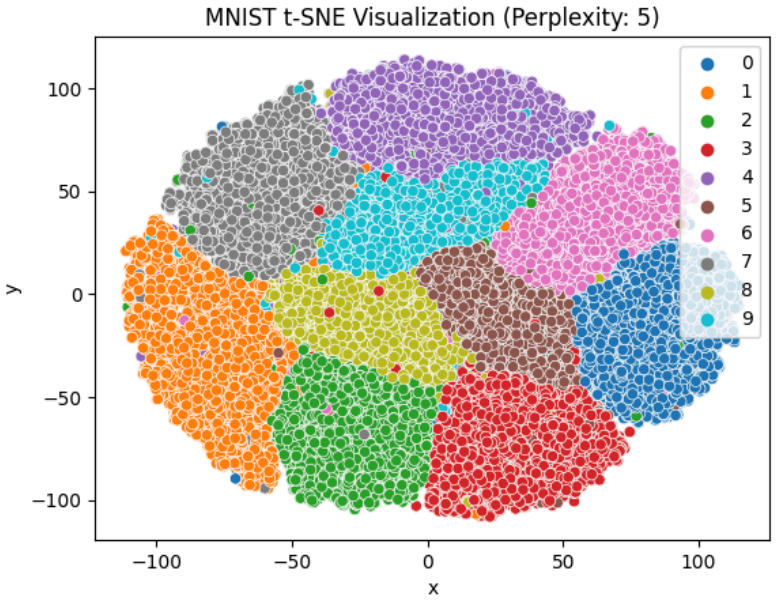
tSNE, Feature Selection, Image HAAR Features

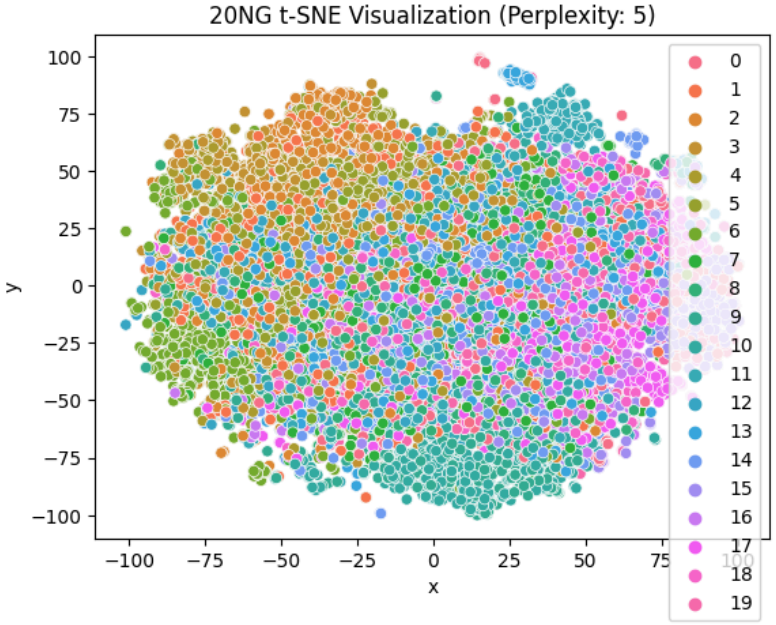
Yusuf Abdi

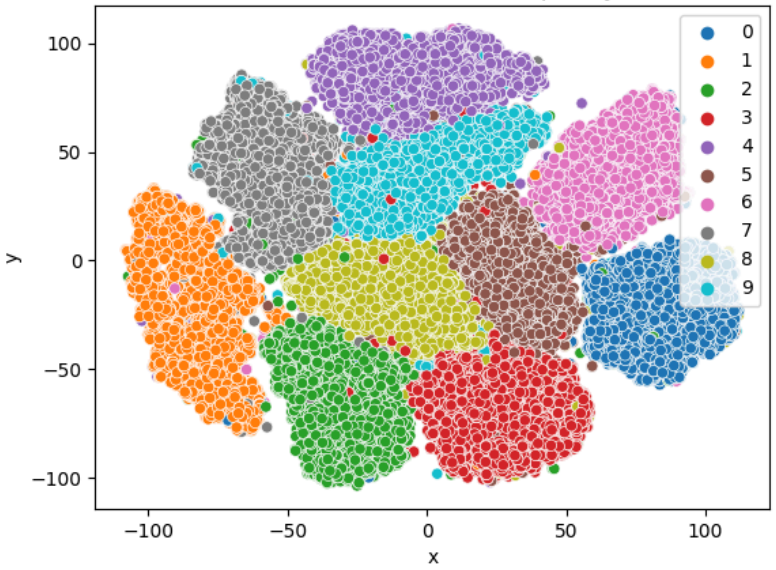
07/07/2023

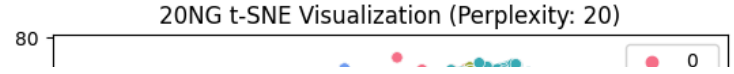
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.manifold import TSNE
from sklearn.datasets import fetch_openml, fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA
import pandas as pd
import tqdm
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.feature_selection import mutual_info_classif
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
import warnings

warnings.filterwarnings('ignore')


# Load MNIST dataset
mnist = fetch_openml('mnist_784', version=1, cache=True)

# Load 20NG dataset
twenty_ng = fetch_20newsgroups(subset='all', shuffle=True, remove=('headers', 'footers', 'quotes'))


# Preprocess the 20NG dataset
vectorizer = TfidfVectorizer(max_features=5000)
tfidf = vectorizer.fit_transform(twenty_ng.data)


# Apply PCA to MNIST dataset
pca_mnist = PCA(n_components=50)  # Adjust the number of components as desired
mnist_pca = pca_mnist.fit_transform(mnist.data)

# Apply PCA to 20NG dataset
pca_20ng = PCA(n_components=50)  # Adjust the number of components as desired
ng_pca = pca_20ng.fit_transform(tfidf.toarray())


# Function to plot t-SNE embeddings
def plot_tsne(embeddings, labels, perplexity, setname):
    df = pd.DataFrame()
    df['x'] = embeddings[:, 0]
    df['y'] = embeddings[:, 1]
    df['labels'] = labels
    df['labels'] = df['labels'].astype('category')
    sns.scatterplot(data=df, x="x", y="y", hue="labels")
    plt.title(f"{setname} t-SNE Visualization (Perplexity: {perplexity})")
    plt.legend(loc='upper right')
    plt.show()

# Perform t-SNE and visualize for different perplexity values
perplexities = [5, 20, 100]

for perplexity in perplexities:
    # Apply t-SNE to MNIST dataset
    tsne_mnist = TSNE(n_components=2, perplexity=perplexity, random_state=42)
    mnist_embeddings = tsne_mnist.fit_transform(mnist_pca)
```

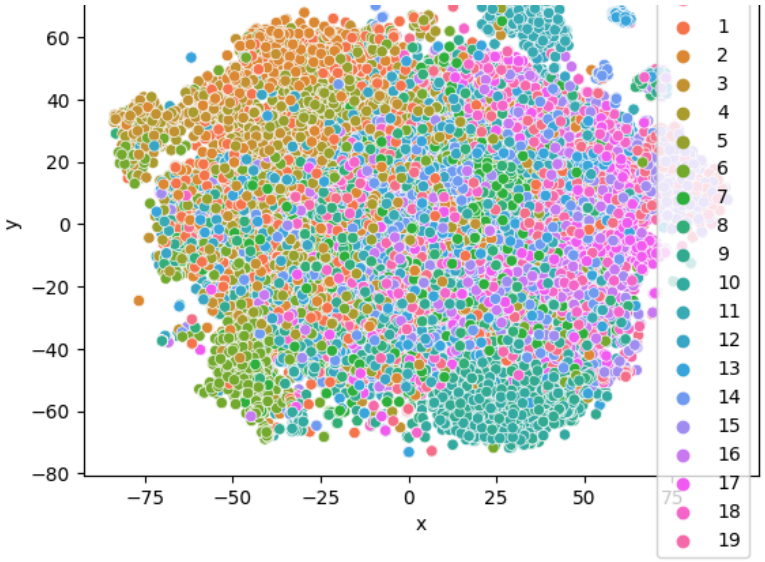
```
# tSNE, Feature Selection, Image HAAR Features
plot_tsne(mnist_embeddings, mnist.target, perplexity, "MNIST")

# Apply t-SNE to 20NG dataset
tsne_20ng = TSNE(n_components=2, perplexity=perplexity, random_state=42)
ng_embeddings = tsne_20ng.fit_transform(ng_pca)

# Visualize 20NG t-SNE embeddings
plot_tsne(ng_embeddings, twenty_ng.target, perplexity, "20NG")
```


MNIST t-SNE Visualization (Perplexity: 5)
x
y
0
1
2
3
4
5
6
7
8
9

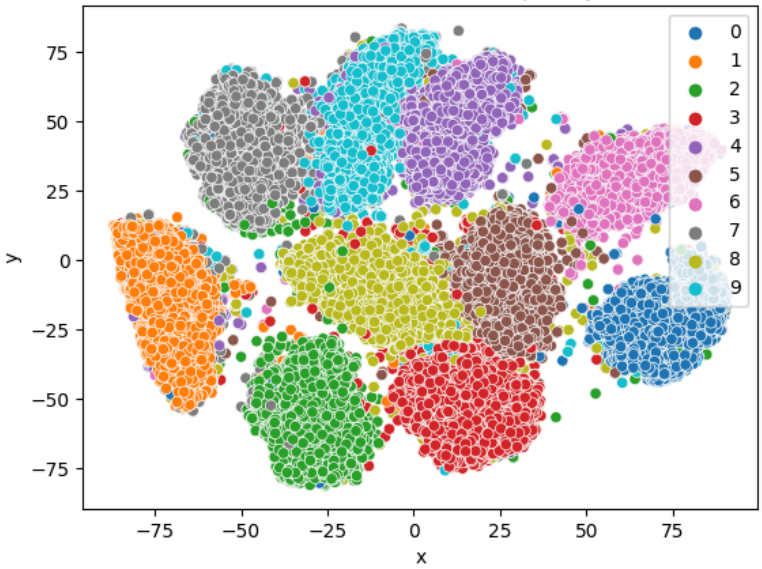

20NG t-SNE Visualization (Perplexity: 5)
x
y


MNIST t-SNE Visualization (Perplexity: 20)
x
y
0
1
2
3
4
5
6
7
8
9

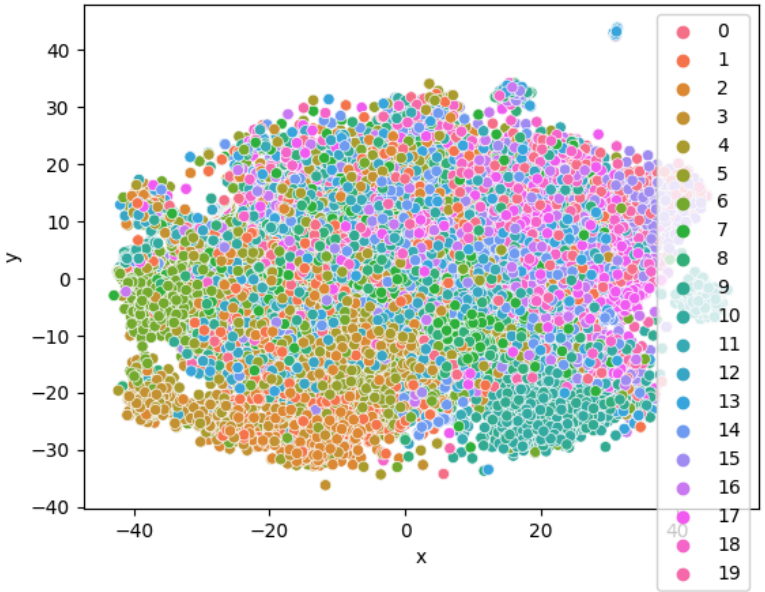

20NG t-SNE Visualization (Perplexity: 20)



MNIST t-SNE Visualization (Perplexity: 100)



20NG t-SNE Visualization (Perplexity: 100)



```
# Load the 20NG dataset
categories = ['alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware',
              'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles',
              'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med',
              'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast',
              'talk.politics.misc', 'talk.religion.misc']

dataset = fetch_20newsgroups(subset='all', categories=categories, shuffle=True, random_state=5)

# Convert text to numerical features using TF-IDF vectorization
vectorizer = TfidfVectorizer(max_df=0.4, min_df=50, stop_words='english', max_features=2000)
X_newsgroups = vectorizer.fit_transform(dataset.data)
y_newsgroups = dataset.target

# Perform feature selection using chi2 criterion
k = 200
selector_chi2 = SelectKBest(chi2, k=k)
X_newsgroups_chi2 = selector_chi2.fit_transform(X_newsgroups, y_newsgroups)

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_newsgroups_chi2, y_newsgroups, test_size=0.2, random_state=42)

# Train a logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.76      0.60      0.67       151
           1       0.70      0.57      0.63       185
           2       0.72      0.64      0.68       200
           3       0.51      0.56      0.53       180
           4       0.77      0.55      0.64       205
           5       0.76      0.67      0.71       213
           6       0.73      0.64      0.68       170
           7       0.84      0.67      0.74       205
           8       0.85      0.82      0.83       208
           9       0.74      0.73      0.73       223
          10       0.86      0.76      0.80       200
          11       0.90      0.81      0.85       199
          12       0.21      0.66      0.32       185
          13       0.80      0.65      0.72       206
          14       0.83      0.83      0.83       187
          15       0.69      0.81      0.75       200
          16       0.71      0.79      0.75       161
          17       0.88      0.81      0.85       192
          18       0.70      0.50      0.58       156
          19       0.74      0.28      0.40       144

    accuracy                           0.67      3770
   macro avg       0.73      0.67      0.68      3770
weighted avg       0.74      0.67      0.69      3770
```

```
# Perform feature selection using mutual-information criterion
selector_mi = SelectKBest(mutual_info_classif, k=k)
X_newsgroups_mi = selector_mi.fit_transform(X_newsgroups, y_newsgroups)

# Split the dataset into train and test sets
X_train_mi, X_test_mi, y_train, y_test = train_test_split(X_newsgroups_mi, y_newsgroups, test_size=0.2, random_state=42)

# Train a logistic regression model
model_mi = LogisticRegression()
model_mi.fit(X_train_mi, y_train)

# Evaluate the model
y_pred_mi = model_mi.predict(X_test_mi)
print(classification_report(y_test, y_pred_mi))
```

```
              precision    recall  f1-score   support

           0       0.39      0.32      0.36       151
```

```
              1       0.30      0.35      0.32       185
              2       0.65      0.55      0.59       200
              3       0.32      0.28      0.30       180
              4       0.32      0.18      0.23       205
              5       0.38      0.42      0.40       213
              6       0.33      0.67      0.44       170
              7       0.28      0.20      0.24       205
              8       0.34      0.40      0.37       208
              9       0.51      0.47      0.49       223
             10       0.46      0.54      0.50       200
             11       0.48      0.48      0.48       199
             12       0.36      0.34      0.35       185
             13       0.36      0.39      0.37       206
             14       0.69      0.66      0.68       187
             15       0.47      0.60      0.53       200
             16       0.33      0.39      0.36       161
             17       0.44      0.60      0.50       192
             18       0.43      0.22      0.29       156
             19       0.47      0.06      0.11       144

       accuracy                           0.41      3770
      macro avg       0.42      0.41      0.40      3770
   weighted avg       0.42      0.41      0.40      3770
```

```
# Perform L1 feature selection using L1-regularized logistic regression
model = LogisticRegression(penalty='l1', solver='liblinear', random_state=42, max_iter=100)
model.fit(X_newsgroups, y_newsgroups)

# Select top 200 features based on the absolute values of the regression coefficients
selector = SelectFromModel(model, threshold=-np.inf, max_features=200)
X_newsgroups_selected = selector.fit_transform(X_newsgroups, y_newsgroups)

# Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_newsgroups_selected, y_newsgroups, test_size=0.2, random_state=42)

# Train a classification model (e.g., logistic regression) on the selected features
classification_model = LogisticRegression()
classification_model.fit(X_train, y_train)

# Evaluate the classification model
y_pred = classification_model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.69      0.63      0.66       151
           1       0.61      0.58      0.59       185
           2       0.73      0.64      0.68       200
           3       0.51      0.56      0.53       180
           4       0.77      0.58      0.66       205
           5       0.74      0.68      0.71       213
           6       0.74      0.74      0.74       170
           7       0.80      0.69      0.74       205
           8       0.82      0.85      0.83       208
           9       0.75      0.76      0.75       223
          10       0.85      0.85      0.85       200
          11       0.87      0.82      0.84       199
          12       0.38      0.54      0.45       185
          13       0.45      0.71      0.55       206
          14       0.80      0.82      0.81       187
          15       0.65      0.74      0.69       200
          16       0.72      0.76      0.74       161
          17       0.87      0.80      0.83       192
          18       0.50      0.51      0.51       156
          19       0.79      0.26      0.40       144

    accuracy                           0.68      3770
   macro avg       0.70      0.67      0.68      3770
weighted avg       0.70      0.68      0.69      3770
```

```
# Function to compute the vertical feature value
def vertical_feature(rectangle, image):
    rows, columns = rectangle.shape
    Q = (rows // 2, 0)  # Point Q on AC
    R = (rows // 2, columns)  # Point R on BD

    ABQR = rectangle[:Q[0]+1, :R[1]]
```