Data Intensitive Computing - HW 1

Introduction

We're going to get started with Predictive Analytics here. The evaluation objectives are to develop a Scikitlearn predictive analytics pipeline and execute visualization using Matplotlib by implementing a predictive analytics algorithm from scratch.

This code snippet imports Python modules needed for data analysis, visualization, and machine learning. Pandas is used for data manipulation, NumPy is used for numerical operations, Seaborn is used for data visualization, and several scikit-learn modules are used for machine learning tasks. It also includes Matplotlib for graphing, Counter for counting occurrences, and SciPy for calculating geographical distances. It appears to prepare the atmosphere for data analysis and categorization activities, but it doesn't appear to do any specific operations.

```
import pandas as pd
import numpy as np
import seaborn as sns
import pickle
import matplotlib.pyplot as plt
from collections import Counter
import scipy.spatial
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
```

- `save_model(model, filename)`: This function is used to save a machine learning model to a file using the `pickle` module. It takes two arguments, the `model` to be saved and the `filename` where the model will be stored.

```
#save the model, and load the same
def save_model(model, filename):
    pickle.dump(model, open(filename, 'wb'))
```

- `load_model(filename)`: This function loads a previously saved model from a file. It takes the `filename` of the saved model as an argument and returns the loaded model. It uses the `pickle` module for deserialization.

```
def load_model(filename):
    with open(filename, 'rb') as file:
        loado = pickle.load(file)
    return loado
"""
```

- `Accuracy(y_true, y_pred)`: This function calculates the accuracy of a classification model by comparing the true labels (`y_true`) with the predicted labels (`y_pred`). It returns the accuracy as a percentage.

```
def Accuracy(y_true,y_pred):
    total = len(y_true)
    correct = np.sum(y_pred == y_true)
    acc = (correct/total) * 100
    return acc
```

- `Recall(y_true, y_pred)`: This function computes the recall, which is a measure of a model's ability to identify all relevant instances. It calculates the ratio of true positives to the sum of true positives and false negatives.

```
def Recall(y_true,y_pred):
   truepos = 0
   falsene = 0
   tar = np.unique(y_true)
   for cls in tar:
```

```
      truepos += np.sum((y_true == cls) & (y_pred == cls))
      falsene += np.sum((y_true == cls) & (y_pred != cls))
  recall = truepos / (truepos + falsene)
  return recall
```

- `Precision(y_true, y_pred)`: Precision measures the ability of a model to identify only the relevant instances. This function calculates the ratio of true positives to the sum of true positives and false positives.

```
def Precision(y_true,y_pred):
    truepos = 0
    falsepos = 0
    tar = np.unique(y_true)
    for cls in tar:
        truepos = truepos +  np.sum((y_true == cls) & (y_pred == cls))
        falsepos =falsepos+ np.sum((y_true != cls) & (y_pred == cls))
    precision = truepos /(truepos + falsepos)
    return precision
```

- `WCSS(Clusters)`: This function calculates the Within-Cluster Sum of Squares (WCSS) for a set of clusters. It measures the compactness of clusters in a K-means clustering algorithm by summing the squared distances of data points within each cluster to their cluster center.

```
def WCSS(Clusters):
    wcss = 0.0
    for cluster in Clusters:
        if len(cluster) == 0:
            continue
        cluster_center = np.mean(cluster, axis=0)
        cluster_distances = np.linalg.norm(cluster - cluster_center, axis=1)
        cluster_wcss = np.sum(cluster_distances ** 2)
        wcss += cluster_wcss
    return wcss
```

- `ConfusionMatrix(y_true, y_pred)`: This function generates a confusion matrix that summarizes the performance of a classification model. It shows the number of true positives, true negatives, false positives, and false negatives.

```
def ConfusionMatrix(y_true,y_pred):
    totc= len(np.unique(y_true))
    cmatrix =np.zeros((totc,totc), dtype=int)
    for i in range(totc):
        for j in range(totc):
```

```
            cmatrix[i, j] = np.sum(np.logical_and(y_true == i, y_pred == j))
    return cmatrix
```

- `KNN(X_train, X_test, Y_train)`: This function implements a k-Nearest Neighbors (k-NN) classification algorithm. It takes training data `X_train`, test data `X_test`, and their corresponding labels `Y_train`. It predicts the labels for the test data using the k-NN algorithm and returns the predicted labels.

```
def KNN(X_train, X_test, Y_train):
   """
   :type X_train: numpy.ndarray
   :type X_test: numpy.ndarray
   :type Y_train: numpy.ndarray

   :rtype: numpy.ndarray
   """
   def distance(X1, X2):
      return scipy.spatial.distance.euclidean(X1, X2)
   k = 7
   # empt arr to store
   Y_pred = []
   for sam in X_test:
      d = []
      votes = []
      for j, train_sample in enumerate(X_train):
         dist = distance(train_sample, sam)
         d.append([dist, j])
      d.sort()
      d = d[:k]
      for dist, j in d:
         votes.append(Y_train[j])
      ans = Counter(votes).most_common(1)[0][0]
      Y_pred.append(ans)
   return np.array(Y_pred)
```

- `Kmeans(X_train, N)`: This function performs K-means clustering on the input data `X_train` with `N` clusters. It returns the cluster centers.

```
def Kmeans(X_train, N):
    """
    :type X_train: numpy.ndarray
    :type N: int
    :rtype: List[numpy.ndarray]
    """

    def distanceo(x,y):
        xsq =np.sum(np.square(x), axis=1)
        ysq= np.sum(np.square(y),axis=1)
        mul =np.dot(x, y.T)
```

```
        dists = np.sqrt(xsq[:, np.newaxis] + ysq - 2 * mul)
        return dists

    def firstc(points, K, **kwargs):
        row,col = points.shape
        ret_arr = np.empty([K, col])
        for number in range(K):
            xrando =np.random.randint(row)
            ret_arr[number] =points[xrando]
        return ret_arr

    def upclu(centers, points):
        row,col = points.shape
        cluster_idx = np.empty([row])
        dist = distanceo(points, centers)
        cluster_idx = np.argmin(dist, axis=1)
        return cluster_idx

    def update_centers(old_centers, cluster_idx, points):
        K, D = old_centers.shape
        new_centers = np.empty(old_centers.shape)
        for i in range(K):
            new_centers[i] = np.mean(points[cluster_idx == i], axis=0)
        return new_centers

    def get_loss(centers, cluster_idx, points):
        dists = distanceo(points, centers)
        loss = 0.0
        N, D = points.shape
        for i in range(N):
            loss = loss + np.square(dists[i][cluster_idx[i]])
        return loss

    def kmeans(points, K, max_iters=100, abs_tol=1e-16, rel_tol=1e-16, verbose=False,
**kwargs):
        centers = firstc(points, K, **kwargs)
        prev_loss = 0.0  # Initialize prev_loss
        for it in range(max_iters):
            cluster_idx = upclu(centers, points)
            centers = update_centers(centers, cluster_idx, points)
            loss = get_loss(centers, cluster_idx, points)
            K = centers.shape[0]
            if it:
                diff = np.abs(prev_loss - loss)
                if diff < abs_tol and diff / prev_loss < rel_tol:
                    break
            prev_loss = loss
            if verbose:
                print('iter %d, loss: %.4f' % (it, loss))
        return centers

    cluster_centers = kmeans(X_train, N)
    return cluster_centers
```

- `SklearnSupervisedLearning(X_train, Y_train, X_test)`: This function trains and tests several supervised learning models, including Support Vector Machines (SVM), Logistic Regression, K-Means clustering, and k-Nearest Neighbors (KNN). It saves the trained models to files and returns their predictions on the test data.

```
def SklearnSupervisedLearning(X_train,Y_train,X_test):
    """
    :type X_train: numpy.ndarray
    :type X_test: numpy.ndarray
    :type Y_train: numpy.ndarray

    :rtype: List[numpy.ndarray]
    """
    """
    :type X_train: numpy.ndarray
    :type X_test: numpy.ndarray
    :type Y_train: numpy.ndarray
    :rtype: List[numpy.ndarray]
    """

    print("SVM Classifier")
    svm_classifier = SVC(kernel='linear',max_iter=1000)
    svm_classifier.fit(X_train, Y_train)
    svm_predictions = svm_classifier.predict(X_test)

    with open("svm_model.pkl", "wb") as svm_file:
        pickle.dump(svm_classifier, svm_file)


    print("Logistic Regression")
    log_reg_classifier = LogisticRegression(max_iter=1000)
    log_reg_classifier.fit(X_train, Y_train)
    log_reg_predictions = log_reg_classifier.predict(X_test)

    with open("logistic_regression_model.pkl", "wb") as log_reg_file:
        pickle.dump(log_reg_classifier, log_reg_file)

    print("K-Means Clustering")
    kmeans = KMeans(n_clusters=len(np.unique(Y_train)))  # Specify the number of clust
ers (adjust as needed)
    kmeans.fit(X_train)
    kmeans_predictions = kmeans.predict(X_test)

    with open("kmeans_model.pkl", "wb") as kmeans_file:
        pickle.dump(kmeans, kmeans_file)


    print("K-Nearest Neighbors Classifier")
    knn_classifier = KNeighborsClassifier()  # Specify the number of neighbors (adjust
as needed)
    knn_classifier.fit(X_train, Y_train)
    knn_predictions = knn_classifier.predict(X_test)
```

```
    with open("knn_model.pkl", "wb") as knn_file:
        pickle.dump(knn_classifier, knn_file)

    return [svm_predictions, log_reg_predictions, kmeans_predictions, knn_predictions]

#
```

- `SklearnVotingClassifier(X_train, Y_train, X_test)`: This function loads pre-trained models (SVM, Logistic Regression, and KNN) and combines their predictions using a Voting Classifier with soft voting. It returns the combined predictions of the ensemble model.

```
def SklearnVotingClassifier(X_train,Y_train,X_test):

    """
    :type X_train: numpy.ndarray
    :type X_test: numpy.ndarray
    :type Y_train: numpy.ndarray

    :rtype: [numpy.ndarray]
    """
    svm_classifier = load_model("svm_model.pkl")

    log_reg_classifier = load_model("logistic_regression_model.pkl")

    knn_classifier = load_model("knn_model.pkl")

    # svm_classifier = SVC(kernel='linear', probability=True)
    # log_reg_classifier = LogisticRegression()
    # knn_classifier = KNeighborsClassifier(n_neighbors=5)  # Adjust as needed
    svm_classifier.probability = True
    voting_classifier = VotingClassifier(
        estimators=[('svm', svm_classifier),
                    ('log_reg', log_reg_classifier),
                    ('knn', knn_classifier)],
        voting='soft'
    )
    voting_classifier.fit(X_train, Y_train)
    voting_predictions = voting_classifier.predict(X_test)

    return voting_predictions
```

---

1. Data is loaded from a CSV file and its summary information is displayed.
2. Outliers are detected using the Z-score method and displayed.

3. The dataset is split into training and testing sets.
4. Data types are converted for compatibility.

```
data=pd.read_csv('data.csv')
data.info()
data.isna().sum()
def outlie(data, th=3):
    zscore = np.abs((data - data.mean( ) ) /data.std())
    out = zscore > th
    return out

out = outlie(data, th=3)
print(out)

# SPlit the dataset into train and test
def split(dataframe, test_size=0.2, random_state=None):
    X = dataframe.iloc[:,:-1 ]
    y = dataframe.iloc[:,-1]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, ran
dom_state=random_state)
    return X_train, X_test, y_train, y_test

X_train, X_test, Y_train, Y_test = split(data, test_size=0.2, random_state=42)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape )
print(Y_test.shape)

#converting d-type cuz i got some error for mismatched d-type
Y_train = Y_train.astype(int)

X_train = np.array(X_train)
X_test = np.array(X_test)
Y_train = np.array(Y_train)
Y_test = np.array(Y_test)
```

The code calculates and prints accuracy, recall, and precision metrics for multiple machine learning models (SVM, Logistic Regression, KNN, K-Means), creates confusion matrices for each model and an ensemble, and generates heatmap visualizations of these matrices for performance evaluation.

```
svm_predictions, log_reg_predictions, kmeans_predictions, knn_predictions = SklearnSup
ervisedLearning(X_train, Y_train, X_test)

def myAccuracyPrecisionRecalls(model_name,Predictions):
    print(f"Accuracy of {model_name} Classifier: ", Accuracy(Y_test,Predictions))
    print(f"Recall of {model_name} Classifier: ", Recall(Y_test,Predictions))
    print(f"Precision of {model_name} Classifier: ", Precision(Y_test,Predictions))
    print("\n")
```

```
myAccuracyPrecisionRecalls("Logistic",log_reg_predictions)
myAccuracyPrecisionRecalls("SVM",svm_predictions)
myAccuracyPrecisionRecalls("KNN",knn_predictions)
myAccuracyPrecisionRecalls("KMeans",kmeans_predictions)

### Using the Scikit-learn library create an ensemble model using the voting classifie
r of the above-mentioned algorithm.

ensemble_voteo = SklearnVotingClassifier(X_train,Y_train,X_test)

myAccuracyPrecisionRecalls("Ensemble Voting",ensemble_voteo)

def plotting_cm(cm, labelling):
    plt.figure(figsize=(8, 6))
    sns.set(font_scale=1.2)
    sns.heatmap(cm, annot=True, xticklabels=np.unique(Y_test), yticklabels=np.unique(Y
_test))
    plt.xlabel('Predicted Values')
    plt.ylabel('True Values')
    plt.title(labelling)
    plt.savefig(f"{labelling}_confusion_matrix.png")


cmEnsemble=ConfusionMatrix(Y_test,ensemble_voteo)
print("Confusion Matrix of Ensemble Voting Classifier:\n ", cmEnsemble)
plotting_cm(cmEnsemble,"Ensemble")

#Confusion Matrix for SKlearn Models: Logistic, SVM, KNN
cmSVM= ConfusionMatrix(Y_test,svm_predictions)
print("Confusion Matrix of SVM Classifier: \n",cmSVM)
plotting_cm(cmSVM,"SVM")

cmLog= ConfusionMatrix(Y_test,log_reg_predictions)
print("Confusion Matrix of Logistic Classifier: \n",cmLog)
plotting_cm(cmLog,"Logistic")

cmKNN= ConfusionMatrix(Y_test,knn_predictions)
print("Confusion Matrix of KNN Classifier: \n",cmKNN)
plotting_cm(cmKNN,"KNN")
```

The code performs the following:

1. Visualizes the error rate for different K values (neighbors) in K-nearest neighbors (KNN) and identifies the optimal K value (in this case, 5 or 7) using an elbow method.
2. Evaluates the optimal number of clusters (K) for K-means clustering, plots an elbow graph, and selects the best K value (in this case, 5).
3. Applies K-means clustering with K=5 to the training data and visualizes the resulting clusters in a scatter plot.

```
### KNN Visualization of Elbow Method
k_values = range(3, 10)
rateOfError=[]
for k in k_values:
    knn_modelling=KNeighborsClassifier(n_neighbors=k)
    knn_modelling.fit(X_train,Y_train)
    rateOfError.append(np.mean(knn_modelling.predict(X_test) != Y_test))
def plot_elbow_graph(modelName, rate_of_error):
    plt.figure(figsize=(10, 6))
    plt.plot(range(3, 10), rate_of_error, marker='o', markersize=5)
    plt.title(f'Rate of Error vs. K Value for: {modelName}')
    plt.xlabel('K')
    plt.ylabel('Rate of Error')
    plt.savefig(f"{modelName}_Elbow.png")
plot_elbow_graph("KNN",rateOfError)


#Optimal K = 5 or 7, which are near the elbow of the graph: Now we update our knn impl
ementation with this value of K
### KMeans Optimal K evaluation
def perform_kmeans(X, num_clusters):
    kmeans = KMeans(n_clusters=num_clusters, random_state=0)
    cluster_assignments = kmeans.fit_predict(X)
    cluster_centers = kmeans.cluster_centers_
    return cluster_centers, cluster_assignments

def visualize_clusters(X, cluster_centers, cluster_assignments):
    plt.figure(figsize=(10, 6))
    plt.scatter(X[:, 20], X[:, 25], c=cluster_assignments, cmap='viridis', marker='o',
s=50)
    plt.scatter(cluster_centers[:, 20], cluster_centers[:, 25], c='red', marker='x', s
=200, label='Centroids')
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('K-means Clustering')
    plt.legend()
    plt.grid(True)
    plt.savefig("Clusters.png")

def optimal_KMean():
    wcssSKLearnImplementation = []
    for k in range(1, 10):
        kmeans = KMeans(n_clusters=k, random_state=0)
        kmeans.fit(X_train)
        wcssSKLearnImplementation.append(kmeans.inertia_)
    return wcssSKLearnImplementation


wcssVals = optimal_KMean()

def identifyOptimalK(wcssVals):
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, 10), wcssVals, marker='o', linestyle='-')
    plt.title("KMeans Elbow")
    plt.xlabel('Number of Clusters (K)')
```

```
    plt.ylabel('Within Cluster Sum of Squares (WCSS)')
    plt.savefig("Kmeans_Elbow_graph.png")

identifyOptimalK(wcssVals)


# Best K Clusters = 5
cluster_centers, cluster_assignments = perform_kmeans(X_train, 5)
visualize_clusters(X_train, cluster_centers, cluster_assignments)
```

The code visualizes decision boundaries of a Support Vector Machine (SVM) classifier for two features, using an SVM model trained on the selected features, and plots the decision boundaries in a scatter plot.

```
### SVM Boundary Visualization for two features
svmFeatures = [10,11,12,13,14]
svmCols= data.iloc[:, [48] +svmFeatures]
svmTarget = [x if x in [0, 1, 2] else None for x in data.iloc[:, -1]]
svmTarget_df = pd.DataFrame(svmTarget, columns=[48])
dataSVM = pd.concat([svmCols, svmTarget_df], axis=1)
dataSVM = dataSVM.reset_index(drop=True)
dataSVM.dropna(inplace=True)
X_svm = dataSVM.iloc[:, :-1]
Y_svm = dataSVM.iloc[:, -1]
xtrainSVM, xtestSVM, ytrainSVM, ytestSVM = train_test_split(
        X_svm, Y_svm, test_size=0.2, random_state=42)


xtrainSVM=xtrainSVM.to_numpy()[:,[0,1]]
ytrainSVM
svmModelo=SVC(kernel='rbf')
svmModelo.fit(xtrainSVM,ytrainSVM)

a,b = -2, 2
c,d = -0.5, 0.5
xx, yy = np.meshgrid(np.arange(a,b, 0.3),
                     np.arange(c,d, 0.3))
preds = svmModelo.predict(np.c_[xx.ravel(), yy.ravel()])
preds = preds.reshape(xx.shape)
plt.figure(figsize=(10, 8))
plt.contourf(xx, yy, preds, alpha=0.8)
plt.scatter(xtrainSVM[:, 0], xtrainSVM[:, 1],
            c=ytrainSVM, cmap=plt.cm.Paired)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title('SVM Decision Boundaries')
plt.savefig("SVM_Boundary.png")
```

The code visualizes the cross-validated accuracy of Logistic Regression models with different hyperparameter combinations (C and max_iter) and plots the trajectory of

hyperparameter settings against accuracy in a bar chart.

```
### Hyperparameter Visualizations
#n Lets visualize the hyperparameter trajectory with different combinations.
logistic_hyperparams = [
    {'C': 0.01, 'max_iter': 10000},
    {'C': 1.0, 'max_iter': 50000},
    {'C': 0.1, 'max_iter': 20000},
]
scores = []
hyperparam_labels = []
for params in logistic_hyperparams:
    clf = LogisticRegression(**params)
    scores.append(np.mean(cross_val_score(clf, X_train, Y_train, cv=5)))
    hyperparam_labels.append(str(params))


plt.figure(figsize=(10, 6))
plt.plot(hyperparam_labels, scores, marker='o')
plt.title('Hyperparameter Trajectory Plot')
plt.xlabel('Hyperparameter Settings')
plt.ylabel('Cross-Validated Accuracy')
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.savefig("Hyperparameter_Trajectory_Plot.png")
```

References

1. Scikit-Learn. (n.d.). Ensembles: Gradient boosting, random forests, bagging, voting, stacking. Retrieved October 4, 2023, from https://scikit-learn.org/stable/modules/ensemble.html
2. Arora, A. (2021, July 24). K-means for beginners: How to build from scratch in Python. Analytics Arora. https://analyticsarora.com/k-means-for-beginners-how-to-build-from-scratch-in-python/
3. Datatofish.com. (n.d.). Example of confusion matrix in Python. Retrieved October 2, 2023, from https://datatofish.com/confusion-matrix-python/
4. Maklin, C. (2018, December 28). K-means Clustering Python example. Towards Data Science. https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203
5. Matplotlib.org. (n.d.). Matplotlib documentation — Matplotlib 3.8.0 documentation. Retrieved October 4, 2023, from https://matplotlib.org/stable/index.html

6. Pydata.org. (n.d.). User guide and tutorial — seaborn 0.13.0 documentation. Retrieved October 4, 2023, from https://seaborn.pydata.org/tutorial.html
7. Wikipedia, The Free Encyclopedia. (2023, August 25). Ensemble learning. https://en.wikipedia.org/wiki/Ensemble_learning