FINAL PROJECT REPORT (COMPETITIVE PROJECT)

CS 6350 – MACHINE LEARNING

HARSH MAHAJAN(u1413898)

GitHub repo - https://github.com/harsh1399/CS-6350---Kaggle-Competition.git

The report contains four sections –

1. Problem definition and motivation
2. Solution
3. Experimental Results
4. Future plan

PROBLEM DEFINITION AND MOTIVATION –

The project aims to predict whether a person earns more than 50,000 USD or less in a year. To make these predictions, we will use the 1994 US Census Data.

I chose the competitive project because, as a student learning Machine Learning, I had heard of Kaggle competitions and always wanted to participate in those competitions. This course and its project have allowed me to start my Kaggle journey. Also, it's exciting and fun to experience a competition.

The training dataset contains 25000 entries. One of the methods to create a classification technique is to manually analyze each of the 25k entries and build a decision construct using if/else statements. The main problem with this would be that it would require a large amount of time to analyze each entry in our dataset. Secondly, it won't be able to classify entries not present in our dataset, i.e., it would not work with a new combination of data. Therefore, I applied the machine learning method to train a model that easily classifies any new data. Also, it is a lot easier to train an ML model than manually constructing a decision tree of if/else statements.

SOLUTION –

- Data Preprocessing –
  Before applying any machine learning algorithm, the first step is understanding the data and various kinds of data validations(ex-null value check). Therefore I started with exploratory data analysis. It was the primary goal of my mid-term report. Few of the conclusions from my mid-term report –
  1. The data consists of thirteen features: seven are categorical, and six are numerical.
  2. Many rows contain missing values, represented by "?". These missing values were replaced by np.NaN. The most occurring value then imputed these NaN values in that column.
  3. We first need to convert categorical features to numerical features to apply the machine learning model, i.e., encode categorical values to discrete numerical values. I was using label encoding initially, but the problem with label encoding was that it introduced ranking in our dataset, although our data was not ordinal.
  4. For numerical features, we applied standardization as it improves model performance.

  I encoded categorical variables using the one-hot encoding technique to implement my final solution. One hot encoding creates new features according to the number of unique values in a

categorical feature. The benefit of using a one-hot encoding technique is that the values in categorical features are not ranked. As our features were not ordinal, implementing one-hot encoding made more sense.

Also, I randomly resampled the dataset to decrease skewness and reach a more balanced distribution between the majority and minority classes. I performed random oversampling, which duplicates examples of the minority class.

- Final Solution –
  I tried implementing various classification algorithms in the mid-term report, such as Decision Tree, Random Forest, Adaboost, and Logistic Regression. As expected, the ensemble method performed much better than the decision tree and logistic regression on the test dataset. The following table shows the result of all four implemented algorithms –

Models	Training Accuracy	Test Accuracy
Decision Tree	99.9%	69.8%
Random Forest	99.9%	77.7%
Adaboost	84%	75.1%
Logistic Regression	80.7%	66.7%

Both Random Forest and Adaboost showed good accuracy on training data but did not perform as expected on the test dataset.

The individual models were not giving me the expected accuracy. Hence, I tried implementing a "Cluster-Then-Predict" methodology that improves the classification accuracy. In this methodology, we cluster our training example using any clustering algorithm and then build a classification model for each cluster[1]. I used the k-means algorithm to cluster the training data.

1. K-Means algorithm –
   K- means is an unsupervised algorithm that creates clusters in data without knowing the output labels. K-means creates k (number of centroids) clusters, and each data point is allocated to a particular nearest cluster. K means algorithm uses Euclidean distance to assign data to each cluster. After each point has been assigned to a cluster, the centroid for the cluster is recalculated. One of the essential things with k-means algorithms is the selection of k, i.e., the number of clusters required. Generally, the elbow method is used to determine the optimal value of k. In elbow method, we vary the number of clusters, and for each value, we calculate WCSS (within-cluster sum of squares). We plot the WCSS value against the number of clusters to find the best value of k. I used the kneed library available on PyPI to calculate the elbow point. It takes a range of clusters and their corresponding wcss distance, and using this wcss distance, it returns the most optimal value of k.

   Using the k-means algorithm, I divided the training dataset into clusters and trained different models on each cluster. The optimal value of k for our training dataset was 4, calculated using the kneed library from PyPI.

```
def optimal_clusters(data):
    wcss_dist=[]
    for i in range (1,8):
        kmeans=KMeans(n_clusters=i,init='k-means++',random_state=10)
        kmeans.fit(data)
        wcss_dist.append(kmeans.inertia_)
    kn = KneeLocator(range(1, 8), wcss_dist, curve='convex', direction='decreasing')
    return kn.knee

no_of_clusters = optimal_clusters(x_sampled)
no_of_clusters
```

```
4
```

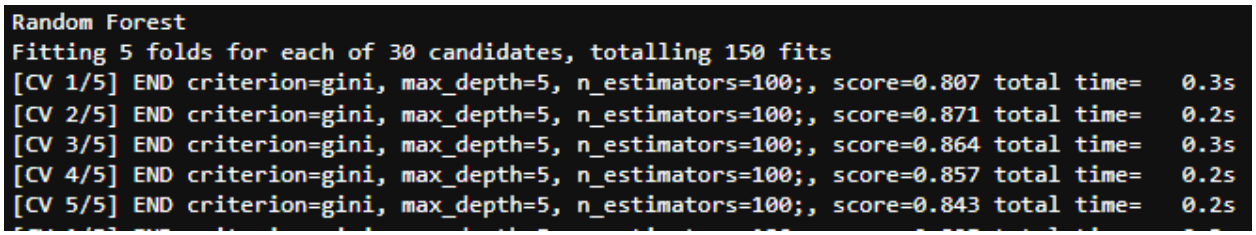
```
kmeans = None
def create_clusters(data,no_of_clusters):
    global kmeans
    kmeans = KMeans(n_clusters=no_of_clusters,init='k-means++',random_state=10)
    cluster_nos = kmeans.fit_predict(data)
    data['cluster_no'] = cluster_nos

create_clusters(x_sampled,no_of_clusters)
print(kmeans)
```

```
KMeans(n_clusters=4, random_state=10)
```

After dividing the data into clusters, the next challenge was finding the ML model that best fits each cluster. For each cluster, I split the data into training and testing datasets, and for each training set, I am implementing a neural network (MLPclassifier), XGBoost (XGBClassifier), Random Forest (RandomForestClassifier), and Naïve Bayes(GaussianNB). I use GridSearchCV to tune parameters for every model and get a parameter set that gives the highest cross-validation accuracy. Then I am using the test dataset to calculate roc_auc_score for each of the trained models and select a model with the highest accuracy for each cluster.

2. GridSearchCV –
   We aim to train different models and select one with the highest performance. One important way to improve the performance of any machine learning model is to tune its hyperparameter at the most optimal value for that dataset. Manually checking different combinations of values for hyperparameters is time-consuming; hence, we use the GridSearchCV library to automate tuning hyperparameters. To pass the different values of hyperparameters, we define a dictionary that contains hyperparameters as keys and a list of values it can take. GridSearchCV tries all the value combinations and evaluates the model using the cross-validation method. Hence after using GridSearchCV, we get an accuracy score for every combination of hyperparameters, and we can use this score to get the best combination of hyperparameters.



3. Neural Network (MLPClassifier)-
   MLPClassifier is a feedforward neural network that can be used for classification problems. The MLPClassifier takes various parameters such as hidden_layer_sizes, activation, solver, max_iter,toI, etc. I experimented with the following values of the above hyperparameters and used GridSearchCV to get the best pair of hyperparameters.
   Hidden_layer_sizes: I tried two different sets of hidden layer sizes. One was two hidden layers, each containing ten neurons, and the other had 15 neurons each.
   Activation - It describes the activation function that needs to be used for each neuron in hidden layers. I used tanh(hyperbolic tan) and relu(rectified linear unit) activation functions.
   Solver – the solver used for weight optimization. I used ADAM(stochastic gradient-based optimizer) and SGD(Stochastic Gradient descent).
   Max_iter: The maximum number of iterations allowed for the neural network. I tried 600 and 700 iterations.

4. XGBoost (XGBClassifier) –
   Extreme Gradient Boosting(XGBoost) is an ensemble technique that implements a gradient boosting algorithm. The main focus of XGBoost is computation speed and model performance. XGBoost is very fast when compared to other implementations of gradient-boosting algorithms. It is one of the go-to algorithms in Kaggle competitions. The two most important parameters for XGBoost algorithms are –
   n_estimators – it specifies the number of boosting rounds. I tried with 100,130 and 150 boosting rounds.
   max_depth – maximum tree depth for weak learners. I tried varying the max_depth attribute between 5 to 10.

5. Random Forest (RandomForestClassifier) –
   Random forest is an ensemble technique that combines various decision trees. It uses the voting technique to combine the results of different decision trees and return a final output. Random forest converts low bias high variance algorithm (Decision tree) to a low bias low variance algorithm. It fits decision trees on various subsets of the dataset and uses averaging to improve accuracy and decrease over-fitting. Essential parameters for random forest classifier –
   Criterion – decides which function to use to determine the quality of the split. I experimented with 'Gini' and 'entropy' values.
   Max_depth – specifies the maximum depth for each decision tree. I tried varying its value from 5 to 10.
   n_estimators: The number of decision trees needs to be used to construct a forest. I tried 100,130, and 150 as estimator values.

6. Naïve Bayes(GaussianNB)-
   Naïve Bayes is a probabilistic model that can be used for the classification task. It is based on Bayes theorem. Bayes theorem gives us the probability of event A happening, given that event B has already occurred. The assumption with naïve bayes is that features are independent i.e., the presence of one feature doesn't affect the other. There are different types of naïve bayes classifiers available but I used gaussian naïve bayes classifier. It implements Gaussian Naïve Bayes algorithms for classification.

As reference [1] suggested, I divided the dataset into clusters using the k-means algorithm. For every cluster, I am finding a model that bests fits the data in that cluster. To predict our test data, we again divide our test data into various clusters using the k-means model we used in our training. Once the test data has been divided into clusters, we use the machine learning model trained for that particular cluster to classify the data.

EXPERIMENTAL RESULTS -

The models that I got for every cluster –

```
[35] cluster_model

    {3: XGBClassifier(criterion='gini', max_depth=9, n_estimators=150),
     1: XGBClassifier(criterion='gini', max_depth=9, n_estimators=150),
     0: XGBClassifier(criterion='gini', max_depth=5, n_estimators=130),
     2: GaussianNB(var_smoothing=1e-08)}
```

I tested the trained model for every cluster on their corresponding training dataset. Naïve Bayes gave the highest accuracy, but it has the least number of points in its cluster. Cluster 3 had the highest set of points in its cluster, and it used the XGBoost algorithm giving 88% accuracy on the training dataset. The following screenshot shows different clusters, their corresponding model, and their accuracy.

```
cluster: 3  model: XGBClassifier(criterion='gini', max_depth=9, n_estimators=150)  training data: 20116  accuracy: 0.88188
cluster: 1  model: XGBClassifier(criterion='gini', max_depth=9, n_estimators=150)  training data: 14969  accuracy: 0.97347
cluster: 0  model: XGBClassifier(criterion='gini', max_depth=5, n_estimators=130)  training data: 2496  accuracy: 0.9935897
cluster: 2  model: GaussianNB(var_smoothing=1e-08)  training data: 387  accuracy: 1.0
```

The final accuracy score of the test data we used was 90.945%.

FUTURE PLAN –

If I had more time, I would use the same methodology that I had used, but I would implement other classification algorithms, such as logistic regression, AdaBoost, SVM, etc. Also, I would implement PCA to reduce data dimension. I would have also tried feature importance using a random forest algorithm and then implemented various classifications using only essential features.

References -

[1] Trivedi, Shubhendu & Pardos, Zachary & Heffernan, Neil. (2015). The Utility of Clustering in Prediction Tasks.