Data Mining Team Project

Introduction and Data Cleaning

Our team was tasked with classifying the income of people found in the Census database. We classified the incomes as above or equal to $50K and below $50K. We ran and compared the popular classification algorithms that we discussed, K Nearest Neighbor (KNN), Naïve Bayes, Decision Trees, and Support Vector Machines (SVM).

We started by cleaning the data to handle imbalanced or missing data. We began cleaning both training and testing data by shaping the dataset into dataframes, renaming column names and deleting irrelevant columns, checking for imbalance and missing value, as well as changing certain dependent variables to binary flag in preparation of further modeling.

Naïve Bayes

Gaussian Naïve Bayes was used for this project since it is one of the most popular and intuitive methods. It is a supervised classification algorithm that makes predicted classifications using conditional probability. This method has been very successful in some real-world situations and tends to work well with even a small amount of training data. We choose to use Naïve Bayes because it works well with categorical variables, like most of those used in our dataset.

Applying Naïve Bayes to the Data

In order to make the data work for the Gaussian Naïve Bayes algorithm provided by sklearn, the categorical data needed to be changed since strings were not accepted. In order to fix this, label encoding was used as our preprocessing technique. The categories 'Work-class', 'Education', 'Marital-status', 'Occupation', 'Relationship', 'Race', 'Sex', 'Native-country', ' <=50K' were changed.

Results

Model	Correctly Classified	Incorrectly Classified	True Positive (above $50K)	False Positive	True Negative (less than $50K)	False Negative
Naïve Bayes	80.27%	19.73%	11,790	645	2,568	1,278

KNN

K Nearest Neighbor (KNN) is a lazy learning algorithm and one of the simplest to implement. Data points are separated into several classes, two in this case, and any new data point is compared to the other data points closest to them. The object is classified by a majority vote of the neighbors. This also makes the prediction sensitive to irrelevant features so the data needed to be cleaned well. We chose to use KNN because it makes no theoretical assumptions about the dataset and has high accuracy.

Applying KNN to the Data

First, we used the information provided in the dataset to label the columns correctly and shaving unneeded punctuation and extra spaces. Then, from the csv file that followed, in the attached KNN.py I utilize missing value imputation by treating the columns with missing values as Y, and the instances with missing values as Test set. Considering the size of the training set compared to the k values, I decided to proceed without a bagging or other weight method, however, the results were shown to be weighted towards recall over specificity, so in the future I would want to evaluate the same problem with more even numbers of Y=1 and Y=0.

After completing imputation, in order to increase the accuracy of the model, I used a method to split up the categorical(non ordinal and non numeric) columns into m binary slices; this is because knn algorithm uses euclidean distances, and thus without this step, there would be weighting according to the ordering of the list of categories. For a similar reasoning, I placed all columns on a normalized scale of [0,1], without this step, numerical columns would outweigh categorical columns easily. Afterwards, I used the KNN algorithm to predict the Test Y based on a model made on Training X and Training Y; this was run over a range of odd k values from 1 to 19. As referenced above, the optimal k for this dataset appeared at k=13.

In evaluating accuracy for the models of various k values, I noticed two interesting things: that the accuracy of different models was extremely close, and that the specificity outweighed the recall by at least 20% on every test model. The first point is interesting because if even a tiny number of instances can accurately predict, then that implies that predictiveness is more definite. This latter point is unusual because normally an unbalanced dataset would have higher accuracy for the majority value, but it is the opposite here; this can likely be attributed to the very high number of total instances compared to the k number which is between 1 and 19. The latter point could be ameliorated with changing the probability threshold needed to evaluate as class one (e.g. greater than a majority is needed to choose class 1 over 0), yet this would result in lower class 1 accuracy as a tradeoff.

Results

k= 13
Training Set:
recall= 0.936205501618123
specificity= 0.7074352761127407
acc= 0.8811154448573447

precision= 0.936205501618123

_____

Testing Set:
recall= 0.9252110977080821
specificity= 0.6664066562662506
acc= 0.8640746882869602
precision= 0.9252110977080821

Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving **regression and classification problems** too. The goal of using a Decision Tree is to create a training model that can be used to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data(training data). In Decision Trees, for predicting a class label we traverse through the decision tree built from the training data (decision Rules), we compare the values of root nodes to the test data point, on the basis of comparison we traverse through the tree till we end up at our leaf node which will be either of the classes.
We chose Decision Tree for the problem at hand as we have to finally classify each record into two classes, which makes it a binary classification problem. Also the interpretation of this is quite easy and more useful than other linear algebra based models like logistic regression. Since Decision Tree works like nested if-else conditions, this makes the model interpretation easy and we understand why we get the result we are getting.

Applying Decision tree on the data

Before applying the decision tree algorithm to the data, we first clean the data in which we convert all the categorical columns to one hot encoded features, and we normalize all numerical columns. After doing the feature engineering the data is ready for the modelling. We first perform 10 fold cross validation, to find the best max_depth(depth of the tree) and min_sample_split (min number of samples that must be present for a split to happen). We select our hyperparameters in such a way that the difference between the accuracies of train set and cv set is least , because if you have a high difference in accuracy between the train and cv that implies the model is overfitting . Then we train our model with the selected hyperparameters. After training the model on the training data , we predict on the test data.

Results

Train Accuracy	CV Accuracy	Test Accuracy
82.78	80.78	74.72

SVM

Support Vector Machines (SVM) is a supervised classification machine learning model. SVM creates a separation line that will divide the data points into two groups, the best classification lines will correctly classify all (or most) of the training data points and leave the largest margin on both sides of the line so that new data points have a better chance of being correctly classified. If the data is not clearly separable then SVM can increase the planes to make the model two dimensional instead of one dimensional, allowing the data to be separable.

We chose to use SVM as one of our classification models because it is one of the most widely used clustering algorithms in industry settings. The ability to use multiple planes also allows for better clustering than algorithms that one use one plane. Some weaknesses of SVM are that it can only be used for two-class tasks, the parameters of a solved model are difficult to interpret, and requires full labeling of input data. The data set that we are using in this model is fully labeled and we only need to classify into two-classes making SVM a reasonable model to use on this data.

Applying SVM to the Data

We used Weka to run SVM on the test data set. We used ten-cross fold validation to run the models so that the test set worked as the training and the test data. We changed variables and created 4 different models for the data, two of the models ran with PolyKernel and two ran with RBFKernel. We changed the exponent used in the PolyKernel model to reflect the number of dimensional planes that the data is transformed to. The higher the plane the better the data can be fit and classified, however, there also runs the risk of overfitting the data. We only tested planes one and two, because plane two had a decrease in the amount of correctly classified instances we surmised that higher planes would continue to decrease. In the RBFKernal, gamma determines the spread of the kernel. When gamma is low the curve of the boundary is low and so the decision range is very broad. By increasing gamma, we tighten the boundary and narrow the decision range narrowing in the field of accurate classifications.

Results

Model	Correctly Classified	Incorrectly Classified	True Positive (above $50K)	False Positive	True Negative (less than $50K)	False Negative
PolyKernel exponent 1	85.11%	14.89%	11675	1664	2182	760
PolyKernel exponent 2	84.62%	15.38%	11491	1560	2286	944
RBFKernel	83.69%	16.31%	11649	1869	1977	786

Gamma .01						
RBFKernel Gamma 1	82.83%	17.17%	11648	2009	1837	787

Anomalies

We would have expected the PolyKernel with exponent 2 to be more accurate than exponent 1. Normally the higher the plane the more accurate the model fits the data to the point of overfitting. However exponent 2 was overall lower in correctly classifying the data, though it was better with identifying the less than $50K than the model built with exponent 1, but model exponent 1 was better at identifying the above $50K instances.

Conclusion

We found that KNN had the most accurate classification when K =13. This 86% accuracy was the highest that we saw through all of the models, with SVM coming in at a close second with 85%. We did not expect KNN to produce the highest accuracy because it is one of the simpler classification algorithms. However, with KNN we could increase the amount of neighbors that each data point is compared to and help increase the accuracy of the classifications.

All of the algorithms that we used on this data set resulted in classification accuracies within 10% of each other. Any of these algorithms could be used in a real-world scenario and produce accurate results. In this case our top algorithm was KNN, but the algorithm that would be chosen in a real-world scenario would depend on multiple factors including variable type, data set size, how much of the data needs to be cleaned, and if you want to use code or tools.