EECS 4404: Machine Learning and Pattern Recognition

Implementation and Comparison of Neural Network, SVC and Random Forest Classifiers for the Adult Dataset

Prepared By: Paul Pronio, Ruizhe Liu, Jaffarjeet Singh Brar, Christopher Boyd, Edward Cong, Basil Mahmood, Hassan Bajwa, Elie Lithwick, Olga Klushina, Priyank Patel

12-18-2020

Introduction

The strategy of our team was to work on the project once most of our other courses are done. That allowed us to focus better and deliver results in a timely manner.

The first stage to our implementation of a classification system was to select an appropriate dataset from the UCI Machine Learning Repository [1]. Several datasets were investigated and ultimately the Adult Dataset [2] was selected. The Adult Dataset is a classification of individuals that either earn more than $50,000 or less than or equal to $50,000 in a year with 14 attributes that can be used to train a classification system. The primary factor contributing to our usage of the Adult Dataset was the large number of data instances available, nearly 49,000.

Having determined a dataset, we turned our attention towards the types of classification systems to implement. The type of classification system also led to the formation of sub-groups as there were several individuals with strong preferences for each of the chosen classification systems. In particular, we selected to implement:

Neural Networks - Paul Pronio, Ruizhe Liu, Jaffarjeet Singh Brar and Christopher Boyd
Supervised Vector Classifier (SVC) - Edward Cong, Basil Mahmood and Hassan Bajwa
Random Forest - Elie Lithwick, Olga Klushina and Priyank Patel

The estimated timeline for the project is in Figure 1. Our group stayed on track and was able to complete the project in four days.

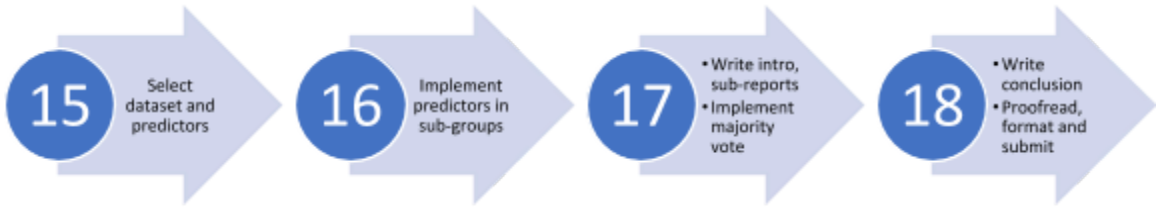

Figure 1: Project timeline - December.

There were no delays or conflicts that arose. Everyone in the group got along very well and we were quick to coordinate and execute the project. The only problem we had was when combining predictors to perform a majority vote, we realized everyone's data was randomized and balanced differently. To fix this issue we balanced and randomized the data set to all the predictors first. That way each group worked with the same data with common indexes to allow for consistent prediction vectors.

Sub-Group 1: Neural Network

This sub-group chose to work with neural networks and consisted of 4 members. We selected neural networks as all of us had an interest in testing them outside of the theory we learned in class. The particular data set we selected seemed to be a good all-around candidate for data processing with all of the different predictor choices. In the meeting with the main group, we were informed of the tool kits that work in python and that they may be easier for us to understand over what was available in Matlab, as initially planned. The libraries we decided on using were sklearn [3] and imblearn [4]. Both of these worked well to get us to a well-trained predictor.

The basic idea is to train a neural network model through the sklearn library [3] with the UCI Adult dataset [2], the training process is as follows:

- **Load the data**: using pandas [5] to load the adult.data and add the corresponding column header.
- **Correlation analysis**: we did a simple correlation analysis to observe the relevance of the features and aimed to drop the features with a correlation score below 0.1, as a final result, we chose to drop five features: fnlwgt, native-country, workclass, education, occupation.
- **Data clean**: rows with missing values were dropped.
- **Map features to numerical value:** map the non-numerical values by categorizing the feature value, e.g 'Married-AF-spouse', 'Married-civ-spouse' will be considered as ''Couple' and mapped to value 1.
- **Split the training set and test set through sklearn:** 20% test data
- **Subsample the imbalanced data through under-sampling**: As the dataset is imbalanced, we chose to balance both the training set and the test set by under-sample through the RandomUnderSampler from imblearn library [4], the amount of training data is reduced from 26048 to 12406.
- **Normalization**: as the Neural Network is sensitive to feature scaling, we normalized the features to improve performance.
- **Train the model with the MLPClassifier**: For the hyperparameter, we chose to use one hidden layer and 9 neurons for each layer as we have 9 features.
- **Evaluation**

  The overall training performance was 81-83%, which was better than the baseline performance of 75%. We used the DummyClassifier function from sklearn library to estimate the baseline on the original dataset. Since the adult dataset was quite large, we chose not to use cross-validation. Instead, the train-test split method was used to improve the accuracy score and determine the final feature subset for better performance. Dropping additional features or adding any of the dropped features such as native-country resulted in a performance accuracy of less than 80%.

F1-score:	**0.82**			
Accuracy:	**0.82**			
	Precision	**Recall**	**F1-score**	**Support**
0	0.84	0.76	0.79	1541
1	0.78	0.86	0.82	1596

Table 1: Neural network classification matrix with F1-score and accuracy.

Sub-Group 2: Support Vector Classifier (SVC)

Our sub-group chose to explore the Support Vector Classifier (SVC) using the python scikit-learn library [3]. In addition to that, we are using the pandas library [5] for data manipulation as well as the NumPy library [6] which is ideal for working with multidimensional data as in the case of the Adult dataset. In regard to training the data, our process is defined below:

- **Load the dataset** using pandas' read_csv function. We had an issue where the dataset was being loaded with an extra space for each feature, we fixed this by passing a skipinitialspace flag to the read_csv function.

- **Convert the output** (value for "over-50K") into a binary value from categorical using pandas' .replace function.
- **Convert the features** from categorical values into binary ones using pandas' get_dummies() function.
- **Correlation matrix:** Generate using pandas' corr() function, use correlation matrix to find all of the relevant features to use. We define a feature as "relevant" if it has more than a 0.25 absolute correlation value.
- **Split data** into training and test data (with the first 80% being training and rest being test). The training data is then shuffled using pandas .sample() function
- **Generate** the X matrix and y vector for both test and training data. The X matrix is the inputs, which we get by dropping every feature not deemed as "relevant" by step 4. The y vector is the output, which we get by only using the "over-50K" column
- **Classify** using a kernelized SVC. We are using a polynomial kernel with a degree of 3.
- **Fit the classifier** to our data by passing in the X and y training values
- Generate a classification matrix by using our classifier to predict outputs from our test X matrix and comparing that with our test y vector

We received an f1 score of approximately 0.77 for <=50,000 (0) and 0.79 for >50,000 (1), using nine features that had an absolute correlation above 0.25. Those features include: 'age' , 'education-num' , 'hours-per-week' , 'marital-status_Married-civ-spouse' , 'marital-status_Never-married' , 'relationship_Husband' , 'relationship_Own-child' , 'sex_Female' , 'sex_Male'.
Our results are summarized in Table 2.

F1-score: **Accuracy:**	**0.79** **0.78**			
	Precision	**Recall**	**F1-score**	**Support**
0	0.79	0.76	0.77	1541
1	0.78	0.80	0.79	1596

Table 2: SVC classification matrix with F1-score and accuracy.

Sub-Group 3: Random Forest Classifier

Our sub-group chose Random Forest Predictor to work with and trained it using the python scikit-learn library [3]. Random Forest predictor is a learning algorithm that uses multiple decision trees and ensembles them to predict the final outcome of the model. For classification problems each decision tree will give us a specific class-prediction and the class with frequent predictions/votes will be our final prediction.
Random Forest is good for a binary classifier because it has the following advantages:

- All the trees are uncorrelated due to splitting of random subsets of features and each tree will consider only a subset of features to predict the outcome which improves bagging.
- The errors of an individual tree won't affect the result, because a group vote considers all the trees.

Our process:

- **Load data**: we loaded the Adult dataset from UCI [2], output was converted to binary and the dataset was balanced.
- **Data analysis**: each datapoint had 15 features/data types in it. We plotted all data that was represented by integers to understand the set better.

- **Correlation matrix**: The result of the correlation matrix showed that Education had the strongest correlation with income, Final Weights had negative correlation so the feature was dropped.
- **Data preparation**: Age and Hours per Week features of data were split into three bins. Most data points had white and US values for Race and Country respectively. That is why we made two bins for both features as white or US and other. Education Number was dropped, because it represented the same information as education. The Education of grades preschool-12 were all grouped together into school as they don't have a degree so can be considered the same education bin. Since Capital Gain and Loss were left skewed, the difference between them was considered as it is more relevant to the Income. The rest of the features were checked for complete data and data points were dropped if any needed values were not present.
- **Data split**: the data was split into three parts: training - 60%, validation - 20% and testing - 20%.
- **Hyper parameters tuning**: Random Forest has a lot of hyperparameters that could be changed, such as the number of trees, maximum number of nodes or number of features per node. We decided to focus on two parameters: max_depth - maximum depth of a tree, min_sample_leaf - minimum number of samples a leaf node must have. These parameters were picked because they prevent the overfitting of our predictor. Predictor was trained with each parameter and validated on the validation set. It can be seen in Table 3, the 10 for min_sample_leaf performed the best. As for the max_depth 50, 100 and 'None' settings performed the same, we chose to use 50 to regularize our hyperparameter and avoid overfitting.

min_sample_leaf	F1- score	Accuracy
1	0.80	79.27
2	0.82	81.31
5	0.82	81.35
10	0.83	81.91
20	0.83	81.71

max_depth	F1-score	Accuracy
10	0.82	80.43
50	0.83	81.91
100	0.83	81.91
None	0.83	81.91

Table 3: Hyperparameters tuning results.

- **Evaluation**: For evaluation we used F1-score and accuracy. The results for test data can be seen in Table 4.

F1-score: **Accuracy:**	**0.82** **0.81**			
	Precision	**Recall**	**F1-score**	**Support**
0	0.83	0.77	0.80	1541
1	0.79	0.84	0.82	1596

Table 4: Random forest classification matrix with F1-score and accuracy.

Lessons Learned

With the ability to generate a machine learning model within a few lines of code thanks to libraries such as sklearn [3], the real work in machine learning is based in retrieving and cleaning data [7]. During our transition from a basic tutorial introducing the use of the various classifiers available in sklearn [8] to a working prediction model this was quickly proven to be true. Starting with the UCI Adult Dataset [4], which contains over 48,000 data vectors, each with 14 attributes, we found that there were many issues with the base dataset. As mentioned previously, there were many vectors missing individual attributes that had to be dropped. Many of the attributes are strings that must be mapped to integer values. Additionally many of the attributes showed little correlation to the classification of an individual making

over $50,000 or less than or equal to $50,000 which were also dropped. As we were starting with a well known and widely used dataset the data cleaning required was relatively minimal. For a real project, it has become apparent to us that the majority of the work would be in obtaining and cleaning the data. In fact, through this project we have learned that the ubiquitous 80/20 rule has a machine learning application: 80% of your time is spent on data pre-processing with only 20% spent on analysis [9]

Conclusion

In reviewing and comparing each classifier, the group as a whole was intrigued by the consistency of the three classifiers relative to each other. Despite not coordinating on any methods for combining, dropping or mapping features, the classifiers were consistent with an accuracy range of 0.78 to 0.82. The three predictors obtained from different techniques of classification were all compared and precision and the f1-score of all predictors were almost the same within +/-0.03 range. However, the neural network and random tree predictors had the same F1-score: 0.82, which was 0.03 higher than SVC, so they performed slightly better.

With the prediction vectors generated by each of the sub-group's classification systems available it was time to add the majority voting system and produce the final results. However, this introduced a point that was not discussed prior to separating into unique groups: the split of data and ensuring a consistent test set. Each group had unique data splits and, therefore, unique prediction vectors that had no correlation with each other. This required the team to step back and coordinate a planned randomization, balance, and split of the data into consistent datasets of 60% training, 20% validation and 20% testing.

With consistent prediction vectors we implemented a simple majority vote between the three classification systems. Unfortunately, adding a majority vote amongst the three different classifiers did not significantly increase the accuracy beyond any of the individual classifiers. With an F1-score of 0.82, which is equal to the scores of neural networks and random forest predictors. The detailed results of majority vote performance can be found in Table 5.

F1-score: **Accuracy:**	**0.82** **0.81**			
	Precision	**Recall**	**F1-score**	**Support**
0	0.83	0.76	0.80	1541
1	0.79	0.85	0.82	1596

Table 5: Majority vote classification matrix with F1-score and accuracy.

References

[1]"UCI Machine Learning Repository: Data Sets", *Archive.ics.uci.edu*, 2020. [Online]. Available: https://archive.ics.uci.edu/ml/datasets.php. [Accessed: 19- Dec- 2020]

[2]R. Kohavi and B. Becker, "UCI Machine Learning Repository: Adult Data Set", Archive.ics.uci.edu, 1996. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Adult. [Accessed: 19- Dec- 2020]

[3]"scikit-learn: machine learning in Python — scikit-learn 0.23.2 documentation", Scikit-learn.org, 2020. [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 19- Dec- 2020]

[4]"imbalanced-learn API — imbalanced-learn 0.5.0 documentation", Imbalanced-learn.readthedocs.io, 2020. [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/api.html. [Accessed: 19- Dec- 2020]

[5]"pandas - Python Data Analysis Library", Pandas.pydata.org, 2020. [Online]. Available: https://pandas.pydata.org/. [Accessed: 19- Dec- 2020]

[6]"NumPy", Numpy.org, 2020. [Online]. Available: https://numpy.org/. [Accessed: 19- Dec- 2020]

[7]J. Osborne, "Data Cleaning Basics: Best Practices in Dealing with Extreme Scores", Newborn and Infant Nursing Reviews, vol. 10, no. 1, pp. 37-43, 2010 [Online]. Available: https://www.researchgate.net/publication/244872484_Data_Cleaning_Basics_Best_Practices_in_Dealing_with_Extreme_Scores. [Accessed: 19- Dec- 2020]

[8]"Income prediction on UCI adult dataset", Kaggle.com, 2020. [Online]. Available: https://www.kaggle.com/overload10/income-prediction-on-uci-adult-dataset. [Accessed: 19- Dec- 2020]

[9]A. Pant, "Workflow of a Machine Learning Project", Medium, 2019. [Online]. Available: https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94. [Accessed: 19- Dec- 2020]