hw2_problem4

March 23, 2020

1 Homework 2, Problem 4

The goal of this problem is for you to try and classify whether or not an individual is likely to make more or less than 50K per year. Carry out this task. Try at least five machine algorithms, report precision, recall and f1 score on a test set.

For each of the parts, report your preformance in terms not of just numbers but in terms of graphs. When you have training and validation data, please show the curves as the training progresses. You should know when you are overfitting or underfitting. Don't just report bare numbers. **You are free to add implmentation or markdown cells to make your notebook clearer!!**

1.1 Data:

The following dataset was taken from the first dataset repository: http://archive.ics.uci.edu/ml/datasets/Adult

As the original task of the dataset lays out, Please note: * the continuous variable fnlwgt represents final weight, which is the number of units in the target population that the responding unit represents.

1.2 Part 1: Dealing with Missing Values

What should you do about dealing with missing values - do you just drop those rows?

One of the most common problems we come accross in working with data "in the wild" is missing data. Often we will have observations (rows) that have only some of the needed attributes. Different rows will have different attributes missing. There are a number of strategies for dealing with the missing values. Clearly one could be dropping the column (attribute), or row (observation). Unfortuntely if you drop columns you may lose critical information that is helpful for classification and may be present in most (many) of the rows. You can drop rows but if many rows have at least one missing value, you may loose too much data. Do you try to impute (i. e. fill in) the missing data? If so how?

Explain why you chose the strategy you did.

Hint - '?' denotes a missing value.

1.2.1 Some possible strategies for dealing with missing data

1. Whenever there is pleanty of data and very little missing data, you should consider dropping rows and/or columns. This may introduce some bias in the data but again, if the problem is limited to a very few rows or columns, it is easy in training to reproduce.

2. Fill with fixed value using sklearn.impute.SimpleImputer.
    a. 'constant' 0. Rarely a good idea but sometimes, if we can assume that when it is missing it is basically 0, this might be a good idea. For example a data may list number of house fires in a zip code and a missing value just means none.
    b. 'mean' if the data is numeric, the mean is meaningfull.
    c. 'median' may be more sensible if the data is integer or ordered. When the mean and median are very different it is important to understand what a "typical" example might mean. When considering "income", for example, a few large outliers will mess up the mean.
    d. "most_frequent' when you have categorical (nominal) labels, mean and median don't make any sense. Most probable label is what you need to use. This is also known as"mode".
3. sklearn.impute.MissingIndicator: Sometimes the fact that a value is missing, is itself an important indicator. One can create a new feature/attribute that indicates a certain attribute is missing. If you later build a classifier by hand you can explicitly wieght each variable using the missing variable weights so that for that example (row) that attribute won't contribute to the classifier. In a deep neural network it is possible that the network can learn to do that automatically.
4. One can use the sklearn.impute.KNNImputer which will look for rows to fill in the data.
5. Fill with sklearn.impute.IterativeImputer scikit-learn provides a sophisticated imputation strategy. You can read up on this in the documentation, but it will fix on of the columns (attributes), and try to use the other features to predict similar to KNN but more sophisticated.
6. Train a classifier: You can build your own classifier using machine learning. This is kind of a problem within a problem but if done correctly, it has the potential to be more accurate than a simpler method. Of course, if done badly it could be worse.
7. Manually impute the missing values. You may know enough about the problem to build an ad-hoc way to fill in the missing values for each column in a way that makes the most sense. This almost always requires a great deal of domain expertise.

[ ]:
```
# Add your code for filling in the data here. Please end by using the␣
 ↪appropriate method for data filling.
# to show the amount of missing data (which in the end should not be any since␣
 ↪you dropped or filled in data)


import pandas as pd

columns = [
    "age",
    "work_class",
    "fnlwgt",
    "education",
    "education_num",
```

```
    "marital_status",
    "occupation",
    "relationship",
    "race",
    "sex",
    "capital_gain",
    "capital_loss",
    "hours_per_week",
    "native_country",
    "target"
]
df = pd.read_csv("adult.data", names=columns)
for column in df.columns:
    if df[column].dtype == "object":
        df[column] = df[column].str.strip()
```

1.3 Part 2: Train Test Validate Split

Ideally you will split the data and use the train data filling in proceedure for the test data. Because this is expensive you can do experiments initially to see if this matters. Just keep carefully in mind what you will know and what you can't know during the test evaluation. Both sklearn and tensorflow provide facilities for train test split. Take your pick.

At the end of this you should have a train, validate and test split. In the next part you are going to do preliminary testing of your model with your train+validation sets to get some idea of good canditates for hyperparameters. Later you will merge your training and validation set and resplit them up using cross validation to get better estimates for setting hyper-parameters

NOTE: It is very important that you record very carefully any parameters you have for filling in data in step 1. For example if you you build a "fit" using some training data, later you will need to use the this "fit" to transform the data, you can not re-fit on new data. In other words if your "pipline" in training takes the mean of the input to fill in the first column, you need to fill with exactly that number, when you get new data for testing. Don't take the mean of the test data.

[1]:
```
# Fill your solution for a train-test split in here.
```

1.4 Part 3: Build different five different variations sklearn models and a Dummy

You will need to use a baseline classifier. Sklearn has sklearn.dummy.DummyClassifier which you can use as a benchmark for a braindead classifier. Pick 5 classifiers including simple ones like Knn or linear like logistic regression, and sophistocated ones like random forest and svm. Use the training and validation data from above (don't look at the testing data). Get a baseline for performance.

Create bar graphs using different metrics including accuracy, recall, precision and f1 score accross the different algorithms.

[2]:
```
# Get baseline results here with logisic regression and random forest

```

```
# Set up your models here

def model_one():
    pass

def model_two():
    pass

def model_three():
    pass

def model_four():
    pass

def model_five():
    pass


# Perform preliminary evaluations here
```

1.4.1 Preliminary conclusions on your models

Include some graphs and peformance metrics

1.5 Part 4: Cross-validation

We really should have used k-fold (eg. k=5) crossvalidation here, to not only evaluate our five keras/tensorflow models. See how your preliminary results change. Now that we have validation results with uncertainy (+- standard deviation), do your prior conclusion change.

[3]:
```
# Part 4 inplement cross validation here
```

1.5.1 Fill in your Part 4 Conclusion here

Explain what you conclude on your models comparing preliminary results to those after cross-validation

1.6 Part 5: Refining with Regularization

We know that our biggest problem, if our models are flexibile enough, will be overfitting. Please try to regularize your best 2 models to see if you can improve their results. Not all algorithms have regularization but analyze two that do. Make sure you show graph performance has you change the regularization parameters. Look at these questions:

- Try regularizing each of your two best models, does the generalizability increase? of Decrease?
- Is one more sensitive than the other? Why might this happen and why?

- Please try this with all of your features and then with the reduced set of features.
- Report your precision, recall and f1 score on the train and validation sets (no cross validatio yet).
- Next carry out cross validation. Does regularization reduce under or overfitting? Why or why not?

** Hint: Try both L1 or L2 norm for regularization or dropout **

[ ]:
```
# Fill in your code analysis for part 5 here
```

1.6.1 Fill in your part 5 conclusions here

Explain what you conclude from your regularization analysis.

1.7 Overall Conclusion

Conclude with a full report here on what we know now about this problem. How well it does verses baseline, what the best Keras archtecture is, what features should be used, how the data should be cleaned etc.

[ ]:
```

```