Project 1

Big Data

Name: Theodoros Mandilaras
A.M.: cs2.190018
MSc DIT/EKPA | Spring 2019-2020.

Files

The project files are:

- README.md → simple analytics in scala
- AnalyticsPySpark.ipynb → analytics in PySpark
- MLpartJ.ipynb → ML in python J
- MLpartK.ipynb → ML in python k

The main Questions have been implemented on **Jupyter** of python3

Setup

Setup completed with success.

Dataset

Dataset completed with success

Simple Statistics in Scala

All the answers are stored in the README.md with the results

Analytics in PySpark

This Question implemented with the **Jupyter**. The solutions are located in the AnalyticsPySpark.ipynb file with the answers in it.

G

Firstly, I take the salary ranges from the real jobs by selecting the salary range column, and I filter the not null values. Then, I split those ranges to the values, I pick the first one, I filter them, I keep those that are numeric and I convert them to integers. By this way I get rid the records that are not accepted.

Then for computing the median I collect the results and I use the numpy.median function.

I

For calculating the most common bigrams and trigrams in real and fake jobs, similar technique has been followed.

Firstly, I select the description column and I keep the not null values. Then, in a FlatMap I use the nltk.util.ngram module on the splitting description in order to get the bigrams or trigrams, and map and reduce by key in order to collect the same together and count them. Lastly, I sort in descend according to that count.

Machine Learning in Python

J

The code is located at the MLpartJ with the answers.
Using only the telecommuting feature the Naive Bayes gives me the below results. As we can see, unlike the accuracy, the precision, recall and f1-measure are very low.

Classifier	Gaussian Naive Bayes
Accuracy	0.9150
Precision	0.05617
Recall	0.05681
F1-Measure	0.0564

Using the has_company_logo and has_questions features in addition to the telecommuting I build a **Random Forest classifier** and a **Persecptron classifier.** The results are show below:

Classifier	Random Forest	Perceptron
Accuracy	0.9542	0.9525
Precision	0.0	0.0
Recall	0.0	0.0
F1-Measure	0.0	0.0

In contrast with the NaiveBayes classifier the other classifiers have zero results for precision, recall and f1-measure. That may has happened by the unbalanced classes. Also, even the high accuracy is plasmatic because the real jobs classes are way more than the fake jobs.

K

The code is located at the MLpartK with the answers.

Part 1: creating a classifier using the description feature

For handling the description feature which it is a text, I used natural process language method. I created a TF-IDF vectorized with the English stop words, and as tokenizer my self created function stemming_tokenizer which splits into words, lowers them, removes the punctuation, and then feeds a PorterStemmer (because it showed better results than LancasterStemmer). Then, I fit_transform the descriptions of my dataset and I create the vectors for each record in the vtrain variable.

As classifiers, I use

- Stochastic Gradient Descent (SGD)
- Support Vector Machines with RBF as Kernel (SVC)

Classifier	SGD	SVC
Accuracy	0.9689	0.9483
Precision	0.9848	0.0
Recall	0.4180	0.0
F1-Measure	0.5869	0.0

At this point, after printing some statistics I decided to create a new dataset by removing some non_fraudulent records in order to get a more realistic and balance dataset (**undersampling**) and then, to evaluate again. At the beginning the dataset had 4.8% fraudulent and 95.2% non fraudulent. After my **undersampling** I make my new dataset at 33.33% fraudulent and 66.6% non fraudulent. The new results are shown below:

Classifier	SGD	SVC
Accuracy	0.897	0.6841
Precision	0.881	0.0
Recall	0.809	0.0
F1-Measure	0.843	0.0

As we can see the accuracy has been decreased, which I believe is a more realistic evaluation for our models.

Part 2: Feature Engineering

For this part, I did not reuse the TF-IDF vector from the description because such a large vector would have absorbed the effect of other features. So, I decided to do some feature extraction from all the columns of my dataset. The features that I collected are:

Features

From columns with large texts I extracted
- Character length (how many characters the record has)
- Character length without spaces
- Number of words

Those are: Company Profile, Benefits, Requirements and Title.

From column with binary features, I kept their values as they are.
Those are: telecommuting, has_company_logo, has_questions

From salary range column I extracted:
- Minimum
- Maximum
- The Difference

From columns with simple(few words length) and repeated text I created a class **Dictionary** which generates ids for new words or returns the id of the word if it already exists in the **Dictionary.**
This technique used for: location, department, employment_type, required_experience, required_education, industry, function.

At the end of that feature extraction, I have collector 28 features from the dataset.

Evaluation

The classifiers that I used are :

- Stochastic Gradient Descent (SGD)
- Support Vector Machines with RBF as Kernel (SVC)
- Decision Tree Classifier (DT)
- Bagging Classifier: using 20 decision trees (estimators)
- Voting Classifier: which contains:
    - One Stochastic Gradient Descent (SGD)
    - One Support Vector Machines with RBF as Kernel (SVC)
    - One Decision Tree Classifier (DT)

In order to find the best one for handling that problem. The results are the following:

Classifier	SGD	SVC	DT	Bagging	Voting
Accuracy	0.8505	0.9556	0.9666	0.9778	0.9533
Precision	0.0753	1.0	0.5980	0.9324	0.7272
Recall	0.1859	0.0507	0.7209	0.5328	0.0827
F1-Measure	0.1072	0.0965	0.6537	0.6781	0.1486

The best performance was given by the Bagging with 97.7% accuracy cause the high efficient of the Decision Trees to handle that problem.

Undersampling

I extracted the features also from the undersampled dataset that I have already created and reevaluate the classifiers. The results are:

Classifier	SGD	SVC	DT	Bagging	Voting
Accuracy	0.6736	0.6841	0.8379	0.9032	0.7424
Precision	0.507	1.0	0.7375	0.8684	0.8488
Recall	0.478	0.045	0.7872	0.8279	0.2597
F1-Measure	0.492	0.087	0.7615	0.8477	0.3978

Just like before, in the undersampled dataset, our classifiers shows lower accuracy than before and that is a more realistic view.

Once again Bagging got the best performance with 90% accuracy.

THE END