Perceptron

Survey data analysis and prediction
using Scikit-Learn Perceptron model

Vittorio Zampinetti

11/02/2019

1 Data Manipulation

Our dataset is a CSV file containing around 150 survey answers (features) for every interviewed developer (approx. 51k samples). Our goal is to make predictions on the salary of a new developer based on those 150 attributes.

1.1 Target

The first thing to do is to delete the samples of which the result (Salary) is uknown, *unobserved*. Then we extract the y vector (Salary column) with which we will train our model. Since we are not interested in predicting the exact salary, we replace the values of y with 0 if the salary is less than the median, with 1 otherwise. Now the domain of the target vector is binary.

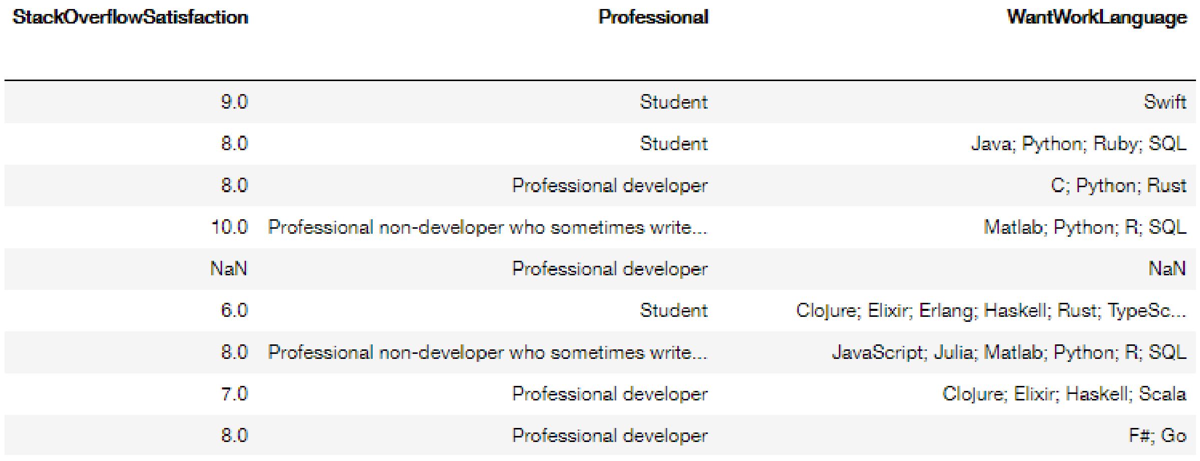
Then we split both the dataframe and the target in train-set and test-set using sklearn `train_test_split()` function.

1.2 Dataset

Then we must transform the dataset (made mostly of string values) with just numerical values since it's the only way to make sklearn algorithms work. To do so we first have to impute unknown values (such as NaN or NULL), replacing it with a constant string (for categories) or with the median of the values (for numeric features).

Then we have to differentiate the features according to their domain; taking a quick look at the dataframe, we see that features can be numerical (e.g. StackOverflowSatisfaction col) or categorical (e.g. Professional col) (see Fig 1). This said we can one-hot-encode those categories, meaning that we replace the column with k binary-values columns where $k = \#categories$. For each sample $k - 1$ values will be zeros, the remaining one (related to the belonging category) will be set to 1.

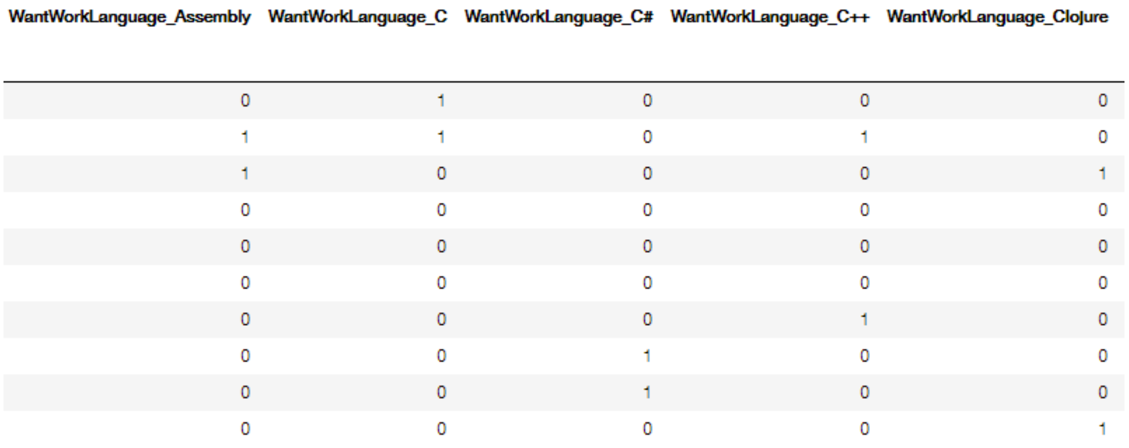
Actually some of the features have multiple-category values (e.g. WantWorkLanguage); one-hot-encoding those features would result in adding a column for each observed combination of categories. Such transformation, besides the fact that the number of columns would increase critically, would not be a smart move. Logically it is more correct to encode



StackOverflowSatisfaction	Professional	WantWorkLanguage
9.0	Student	Swift
8.0	Student	Java; Python; Ruby; SQL
8.0	Professional developer	C; Python; Rust
10.0	Professional non-developer who sometimes write...	Matlab; Python; R; SQL
NaN	Professional developer	NaN
6.0	Student	Clojure; Elixir; Erlang; Haskell; Rust; TypeSc...
8.0	Professional non-developer who sometimes write...	JavaScript; Julia; Matlab; Python; R; SQL
7.0	Professional developer	Clojure; Elixir; Haskell; Scala
8.0	Professional developer	F#; Go

Figure 1: Domains

the features setting 1 for each category that appears in the combination, 0 for the others. To do so we define a function `multi_cat_transform(X)` which takes the multi-category columns and, using a specific Pandas function (`str.get_dummies(sep='; ')`), splits them in encoded unique categories as shown in figure 2 (in the second row both *Assembly*, *C*



WantWorkLanguage_Assembly	WantWorkLanguage_C	WantWorkLanguage_C#	WantWorkLanguage_C++	WantWorkLanguage_Clojure
0	1	0	0	0
1	1	0	1	0
1	0	0	0	1
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	0	0
0	0	0	0	1

Figure 2: Multiple category feature encoding example

and *C++* are set to 1).

To select all of those particular features we could take every feature one by one and look at the values, or, more easily, search for the columns whose unique-values number is more than a certain threshold (found empirically). High unique values count is probably due to combinations of categories, which are hardly identical.

Finally we just have to scale all the values around zero (basic Perceptron rule), and we do this with sklearn StandardScaler.

We put all these transformations together on the train-set with sklearn Pipeline and ColumnTransformer mechanism and we have our *X_ train* array.

2 10-Fold Cross Validation

Now that we have our train-set prepared we can estimate the accuracy of the Perceptron model performing a 10-fold cross validation with sklearn `cross_val_score()` function. It returns the rate of success for each of the 10 iterations. In our case these are the results, which can be considered acceptable

10-fold cross validation scores:
0.840, 0.786, 0.853, 0.835, 0.846, 0.834, 0.854, 0.838, 0.812, 0.838

Mean: 0.834

3 Grid Search

After that we can look for optimal hyperparameters for our model with a *grid search*: we set the parameters space (which in our case is 2D since we just look for best *tolerance* and best number of *epochs*) and sklearn `GridSearchCV` algorithm will find the parameters that maximize the accuracy of the predictions.

Best parameters: epochs: 20, tol: 0.01

4 Prediction Test

Using the parameters found in the previous step we now train the Perceptron model, fitting it with the train-set. Then we let it make predictions on the test-set taken apart and we compare the results with the y-test, generating a score of 0.850.

5 Learning Curve

Finally we can check if our dataset is actually not too small (nor too big) so to avoid both *underfitting* and *overfitting*. To do this we can plot the learning curve of the Perceptron, running a k-fold cross validation on different sample sizes. Sklearn `learning_curve()` function does this operation and returns the scores both of train-set and test-set. The resulting plot is shown in figure 3. We can see that for sizes greater than 6k samples the accuracy is quite good, and also that with sizes lower than 12k samples it doesn't overfit (otherwise we would have seen the training-score line go down and then back up).

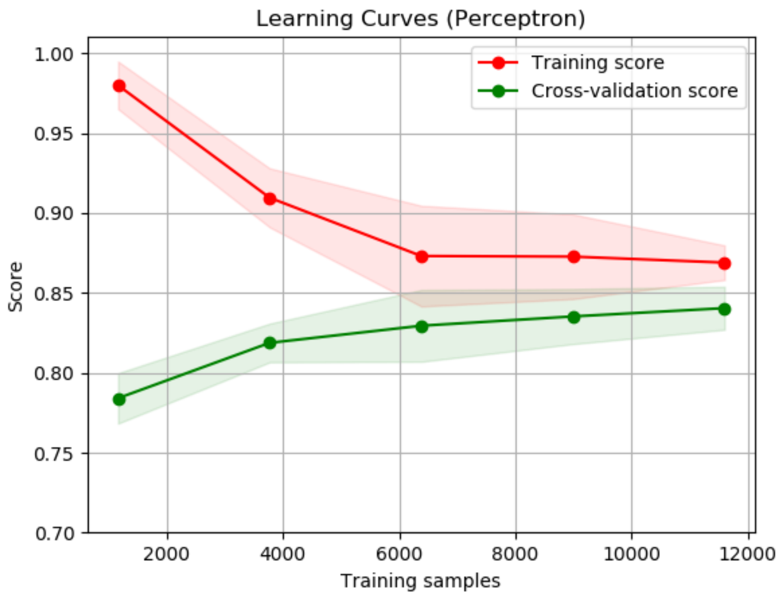


Figure 3: Learning Curve