MODULE 08 - Piscine Python for Data Science

Intro to Machine Learning

Summary: This day will help you with basic tasks of machine learning in Python.

Contents

Chapter I

Foreword

- There are a lot of different terms that are connected to the data field: artificial intelligence, machine learning, neural nets, and deep learning. But do you know the difference between them?
- Each item of the list is a subset of the previous item. The broadest term is artificial intelligence includes any techniques that mimic human cognitive behavior. It can use machine learning algorithms in order to do this, or any other techniques like just writing a program with many rules "if-then-else".
- Machine learning includes statistical algorithms that automate the process of the rules creation. The machine can find correlations and use them for different tasks by "looking" at the data.
- Neural nets are a subset of machine learning algorithms. They were created by the inspiration of how the human brain works (but still it is much far away from it). And deep learning algorithms are a subset of neural nets. They usually have many layers. That is why they are called "deep".
- Remember that all these algorithms are not limitless. They can help you only if you have the data. Simple algorithms can be satisfied with small amounts of data, and the deep learning algorithms require big amounts of data. At the same time, if your data is garbage, the knowledge that you get from the algorithms is garbage too. No magic, huh?

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Here and further we use Python 3 as the only correct version of Python.
- The python files for python exercises (module01, module02, module03) must have a block in the end: if ___name___ == '___main___'.
- Pay attention to the permissions of your files and directories.
- To be assessed your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google.
- Remember to discuss on the Intra Piscine forum.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- And may the Force be with you!

Chapter III

Specific instructions of the day

- Use Jupyter Notebook to work with your code
- For each major subtask in the list of any exercise (black bullets), your ipynb file should have an h2 heading to help your peer easily navigate in your code
- No imports allowed, except those explicitly mentioned in the section “Authorized functions” of the title block of each exercise
- You can use any built-in function, if it is not prohibited in the exercise
- Save and load all the required data in the subfolder data/
- scikit-learn (0.23.1) is the library that you need for all the machine learning tasks.

Chapter IV

Exercice 00 : Binary classifier

	Exercise 00
Binary classifier	
Turn-in directory : *ex*00/	
Files to turn in : `00_binary_classifier_logreg.ipynb`	
Allowed functions : `no restrictions`	

You will work with machine learning during this and the next day. This day will cover the basic stuff in order not to get you overwhelmed by a lot of information. The next day will cover some more sophisticated techniques. So stay tuned!

First of all, machine learning can be different: supervised and unsupervised. In supervised learning, you want to predict something. In order to do this, you give the machine examples: a bunch of features and a target variable. Imagine that you want to predict if a user would like or dislike a given movie. The features (X) can be: genre, year of creation, budget, cast, director, and so on. The target (y) will be like/dislike. That kind of task is a classification task. In classification problems, y is always categorical. If your target variable is continuous (for instance, movie rating), it is called a regression problem.

Unsupervised learning does not require labels (target variable). It does not forecast anything. Usually, it helps to understand your data better. For example, clustering algorithms help you identify homogeneous groups of observations. You may find that you can divide your users into 6 groups. And for each of these groups create a special offer or a recommendation. Do not confuse it with classification. You are not trying to predict anything. You are just looking at your data.

Besides clustering algorithms, unsupervised learning includes dimensionality reduction algorithms. They help you to reduce the number of features or observations. It may be helpful if you have a lot of them, but you do not have sufficient resources. Also, they may help you to find some latent features and improve the quality of your algorithms in supervised learning.

But enough of the theory! Let us try to train our first classifier. It will be a binary classifier which means that the target variable has only two unique values. You will work with the dataset from the previous days. It is more than normal. Actually that is how data science works. You start from the descriptive analysis by analyzing some basic statistics. Then you go further and do explorative analysis by drawing different plots and as a result, understanding your data better. And only then you go to predictive analysis to forecast something.

In this exercise imagine the following situation (that is, by the way, is quite common). From some moment of the past, you realized that the more data you collect, the better. And you started saving more fields in the logs. Before, you had been collecting only the time of the commit. Since that point of time, you started collecting the date too.

Now you need to train a classifier that can predict if any given commit was made in working days or during weekends. Then you will be able to use the classifier to label the commits in the past when you had not been collecting the data.

Every supervised machine learning algorithm requires at least two arguments: X and y. X is the list of features and y is the target column. But what are the features?

As you remember, we have only logs like this 2020-04-17 05:19:02.744528. How can we use it to predict the type of weekday? That is a creative part of the work. It is called feature engineering. You need to extract these features from the logs. What can it be? Remember that the observation in this task is a day? So we need to extract something that can characterize days somehow. What can it be? It may be, for example, the number of commits during the day. It may be the number of commits before midday and after it. Or it may be also the percentage of commits made before midday. Your fantasy is your only limit!

In this exercise, we will try a simple approach with only two features: the number of commits before midday and the number of commits after midday. What you need to do:

- get the data from the checker (only timestamp field) table using sqlite3, the only filter is we need data only of the users and not the admins
- create a dataframe df with the columns: date, am, pm, target, where date is the date of the day, am is the number of the commits during the day before midday (integer), pm is the number of commits during the day after midday (integer), target is weekend or working_day
- create a plot where x is am, y is pm, each dot is a day, working days and weekends must have different colors
- by looking at the graph do you think it will be easy to classify the days having those two features? put your answer in the markdown: yes, it is easy or no, it is not easy.
- train logistic regression on your data using am and pm, parameters are: random state=21, fit_intercept=False
- make predictions for every day of your dataset and add them to your dataframe with the column name predict
- save the dataframe into a file in the subfolder of the day data with the name am_pm.csv
- draw another plot like before, but the color should be taken from the predict
- by looking at the graph do you think if it made good predictions? put your answer in the markdown: yes, it is good; no, it is not good

- calculate accuracy for your predictions
- calculate accuracy for the naive case when each of your prediction is the value of your most popular class of the day
- by comparing the accuracies do you think that the classifier made good predictions? put your answer in the markdown: yes, it is good; no, it is not good

Chapter V

Exercice 01 : Decision boundaries

	Exercise 01
Decision boundaries	
Turn-in directory : *ex*01/	
Files to turn in : `01_binary_decision_boundaries.ipynb`	
Allowed functions : `no restrictions`	

Ok, you trained your first classifier! Now it looks probably like magic to you. Let us look under the hood, try to increase the quality of your classifier, and try other algorithms.

- read the file am_pm.csv to a dataframe
- draw a plot where the x-axis is am, the y-axis is pm, dots are the days, color depends on the target
  - add the decision boundary of logistic regression to the plot
- draw the same plot (with the boundary), but the color should depend this time on the predictions
  - now it should be clear for you how the logistic regression works
- apply StandardScaler to X and train logistic regression with the same parameters again
  - Linear models can be sensitive to the scale of your variables. You make it easier for them to find the optimal solution when you scale your features
  - calculate the accuracy for the new model, did it get better than a naive classifier with the most popular class?
  - draw the plots that were described above but for the new model
- apply to the scaled dataframe SVC model with parameters probability=True, random_state=21

    - calculate accuracy
    - try different kernels, find the best in terms of accuracy
    - draw both plots again with the decisions boundary to see how this algorithm works

- apply to the scaled dataframe DecisionTreeClassifier with parameters max_depth=4, random_state=42
    - calculate accuracy
    - try different values of max_depth
    - draw both plots again with the decisions boundary to see how this algorithm works
    - using method .plot_tree() visualize the decision tree itself
    - how many leaves predict days as working days? put your answer in the markdown

Chapter VI

Exercice 02 : Multiclass

	Exercise 02
Multiclass	
Turn-in directory : *ex*02/	
Files to turn in : `02_multiclassi_one-hot.ipynb`	
Allowed functions : `no restrictions`	

Ok, now you understand more or less how different algorithms make classifications. It is time to go closer to real life. In real life, your target column may contain more than two values. In such cases, you will need to train not a binary classifier, but a multiclass classifier (do not confuse it with multilabel – it is when your observations may belong to several classes at the same time).

Also, you may have not only continuous features but categorical as well. You need to deal with them somehow. Algorithms understand numbers and they do not know what to do with text.

In this exercise, you will work with both problems. Also, you will find out which features seem the most important for different algorithms. You will try one more algorithm as well – random forest.

What you need to do:

- get the data from the checker (uid, labname, numTrials, timestamp) table using sqlite3, the filters are: we need data only of the users and not the admins and also only the rows where status = "ready"
    - create a dataframe df with the columns: uid, labname, numTrials, hour, dayofweek where hour is extracted from the timestamp as well as the dayofweek (0 is Monday, 6 is Sunday), we will try to predict the day of the week having data about which user made a commit for which lab at which hour and which try it was
- using OneHotEncoder() transform your categorical features, delete from the dataframe the initial columns
- use StandardScaler() and scale your continuous features

- save the dataframe as dayofweek.csv
- before trying out different algorithms, find out the accuracy of the naive algorithms – the one that predicts everything as the most popular class
- train logistic regression, for the baseline model use random state=21, fit_intercept=False, calculate the accuracy
    - write a function that draws the plot (barh) taking coefficients of any trained models, names of the features and the number of top-n most important features to display
    - draw a plot (barh) for the baseline model with top-10 most important features (absolute value) for the trained model
    - remember that it is a multiclass classification and coef_ returns a matrix, to calculate importance for a feature you need to sum all the individual feature importances for all the target values
- train SVC model, for the baseline model use parameters kernel='linear', probability=True, random_state=21, try different kernels, calculate the accuracies
    - draw a plot (barh) for the baseline model with top-10 most important features (absolute value) for the trained model for the linear kernel
        - by default SVC uses "one vs one" strategy of the classification, thus in coef_ it returns a matrix
        - to calculate importance for a feature you need to use OneVsRestClassifier over the SVC and sum all the individual feature importances for all the target values
- train DecisionTreeClassifier, for the baseline model use max_depth=4, random_state=21, try different values of max_depth, calculate the accuracies
    - draw a plot (barh) for the baseline model with top-10 most important features (absolute value) for the trained model using the written function
- train RandomForestClassifier, for the baseline model use parameters n_estimators=100, max_depth = 25, random_state=21 try different values of max_depth and n_estimators, calculate the accuracies
    - draw a plot (barh) for the baseline model with top-10 most important features (absolute value) for the trained model using the written function
    - check in the documentation how this model works

Ain't it cool that we can predict the weekday of any commit with high accuracy knowing who made a commit, at what time and for which lab and what number of tries they have already made?

Chapter VII

Exercice 03 : Overfitting

	Exercise 03
Overfitting	
Turn-in directory : *ex*03/	
Files to turn in : `03_split_crossval.ipynb`	
Allowed functions : `no restrictions`	

We are sure that you managed to achieve pretty high values of accuracy. But those numbers are kind of not fair. Why? You were making predictions on the same data that you had used for training. Your models could just memorize all the observations. In theory, you could achieve 100% accuracy, but would your model be still good if it tried to make predictions for the data that it had not seen? We highly doubt it. That is called overfitting.

One of the techniques to prevent it is to make a train/test split. You get a portion of data that you use for training, and another portion you use to check the final quality of your model. The second is cross-validation. In this technique, we do not make a constant split, but we try k different splits and see what quality of predictions we have.

What you need to do:

- read the file dayofweek.csv to a dataframe
- using train_test_split with parameters test_size=0.2, random_state=21 get X_train, y_train, X_test, y_test
- using, for example, value_counts() to check if the distribution of classes is similar in train and test
- use the additional parameter stratify= and check the distribution again, now it should be more or less similar
- train exactly the same baseline models from the previous exercise and calculate the accuracies using the test dataset with stratification
  - did all the models show the similar values of the metric? which one has the largest difference comparing the current exercise and the previous? put the

answer to a markdown cell

- try to play with the parameters of the model, maybe you can achieve higher results
  - is it a good practice to play with the parameters on the test dataset? put the letter of the correct answer to a markdown cell:
    - yes, there is nothing wrong
    - yes, you can achieve better results
    - no, you will overfit model hyperparameters on this test dataset and it will decrease its quality in the production
    - no, you will underfit model because it has to learn something from the new dataset
- using cross_val_score with cv=10 calculate the mean accuracy and standard deviation for every model that you used before (logreg with solver='liblinear', SVC, decision tree, random forest)
- using StratifiedKFold with n_splits=10 calculate the mean accuracy and standard deviation for every model that you used before (cross_val_score with specified cv uses StratifiedKFold by default, but still it is good to learn to do it manually for non-standard use cases)
  - compare the results with the cross-validation without stratification
- choose the best model and play a little bit with the parameters on cross-validation, find a good enough parameter or a combination of the parameters
  - calculate the accuracy for the final model on the test dataset
  - draw a plot that displays the top-10 most important features for that model
  - save the model using joblib
  - load the model, make predictions for the test dataset and calculate the accuracy

Chapter VIII

Exercice 04 : Regression

	Exercise 04
Regression	
Turn-in directory : *ex*04/	
Files to turn in : `04_regression.ipynb`	
Allowed functions : `no restrictions`	

You know now something about classification tasks. Let us work in this exercise on a regression problem. You will need to predict the average delta between deadlines and the first commit for every user having data about their views of the Newsfeed and the number of commits they made during the program.

What you need to do:

- get the data about the average difference of the users from the test and control tables, we ignore project1 this time, between their first commit of a lab and the deadline
- get the data about the page views per user from the pageviews table, we ignore the admins this time
- get the data about the number of commits per user from the checker table, we ignore the admins and project1 this time
- collect all the data into a new dataframe df where uid is the index, and the columns are: num_commits, pageviews, avg(diff)
- fill the missing values in pageviews with 0
- save the dataframe to a file regression.csv
- make a split of your dataset on train and test with parameters test_size=0.2
- write a function crossval that takes as arguments: n_splits for KFold(), X, y, model instantiated class with the parameters of the model (keep in mind: random_state=21) and returns for a given model class a result like this:

```
train - 2696.4496895424836 |  test - 1589.9979527104958
train - 2660.957874001452 |  test - 2075.102636027137
train - 2847.315529246795 |  test - 320.911928168403
train - 2500.7691099659237 |  test - 4132.461382030178
train - 2643.927917295123 |  test - 2237.8140952197878
train - 2396.295678819444 |  test - 4509.650064742476
train - 2003.402267924976 |  test - 8403.491474908551
train - 2531.876094212613 |  test - 3135.944102735099
train - 2683.1795186023123 |  test - 1796.01426292594
train - 2537.1192483996338 |  test - 3439.29824116941
Average RMSE on crossval is 3164.0686140637476
```

- run the function for LinearRegression, DecisionTreeRegressor, RandomForestRegressor
  - you may choose the parameters by yourself, but find the good enough for you
- make predictions for the test dataset using each of the three models with the finalized parameters
  - draw a plot for each of the models where the x-axis is the actual average difference and the y-axis is the prediction made by a model
  - how would it look in the ideal case? put the answer to a markdown cell

Chapter IX

Exercice 05 : Clustering

	Exercise 05
Clustering	
Turn-in directory : *ex*05/	
Files to turn in : `05_clustering.ipynb`	
Allowed functions : `no restrictions`	

It is time to try using unsupervised machine learning. This time we will work with clustering algorithms. We will try to understand if we can divide our users into some homogenous groups for future analysis. Maybe we can add some more triggers or engaging mechanics for them. But the triggers and mechanics may differ depending on the users' existing behavior.

What you need to do:

- read the file regression.csv to a dataframe
- remove the pageviews, we will cluster the users only by the number of the commits and their average difference
- KMeans
  - use this algorithm to create clusters, use random_state=21 and try n_clusters=3
  - visualize the data on a scatter plot
  - try different values of n_clusters and see how your plot will change
  - calculate the silhouette_score
- DBSCAN
  - use this algorithm to create clusters, use eps=20 and try min_samples=2
  - visualize the data on a scatter plot
  - try different values of eps and min_samples see how your plot will change

- calculate the silhouette_score

- AgglomerativeClustering
  - use this algorithm to create clusters and try n_clusters=5
  - visualize the data on a scatter plot
  - try different values of n_clusters and see how your plot will change
  - calculate the silhouette_score
  - visualize the dendrogram

- write a function that:
  - takes as arguments: model class of clustering, its parameters, the name of the parameter for optimization, the range of the parameter values to try
  - tries different values from the given parameter range and calculates the silhouette_score for each value from the range
  - finds out the best value for the parameter in the range
  - returns two subplots:
    - the first shows how the silhouette_score changes depending on the value of the parameter
    - the second visualizes the data on a scatter plot using the clustering model with the best value of the parameter