Introduction to Scikit learn

Yu-Heng, Hsieh

10/24

What is Scikit Learn

- A python library that provides various implementation of machine learning/data mining algorithms
- Clustering
  - SVM, K-means, DBScan,
- Classification
  - NN, decision tree, KNN, naïve bayes

Install Scikit learn

- Use pip to install
- python –m pip install sklearn
- Need other libs like numpy and scipy. Can also install through pip
- If you cannot install scipy on windows you can install the package from http://www.lfd.uci.edu/~gohlke/pythonlibs/#scipy

Datasets

- Scikit learn provides some useful datasets in the "datasets" module.

datasets.fetch_20newsgroups ([data_home, …])	Load the filenames and data from the 20 newsgroups dataset.
datasets.fetch_20newsgroups_vectorized ([…])	Load the 20 newsgroups dataset and transform it into tf-idf vectors.
datasets.fetch_california_housing ([…])	Loader for the California housing dataset from StatLib.
datasets.fetch_covtype ([data_home, …])	Load the covertype dataset, downloading it if necessary.
datasets.fetch_kddcup99 ([subset, data_home, …])	Load and return the kddcup 99 dataset (classification).
datasets.fetch_lfw_pairs ([subset, …])	Loader for the Labeled Faces in the Wild (LFW) pairs dataset
datasets.fetch_lfw_people ([data_home, …])	Loader for the Labeled Faces in the Wild (LFW) people dataset
datasets.fetch_mldata (dataname[, …])	Fetch an mldata.org data set
datasets.fetch_olivetti_faces ([data_home, …])	Loader for the Olivetti faces data-set from AT&T.
datasets.fetch_rcv1 ([data_home, subset, …])	Load the RCV1 multilabel dataset, downloading it if necessary.
datasets.fetch_species_distributions ([…])	Loader for species distribution dataset from Phillips et.
datasets.get_data_home ([data_home])	Return the path of the scikit-learn data dir.
datasets.load_boston ([return_X_y])	Load and return the boston house-prices dataset (regression).
datasets.load_breast_cancer ([return_X_y])	Load and return the breast cancer wisconsin dataset (classification).
datasets.load_diabetes ([return_X_y])	Load and return the diabetes dataset (regression).
datasets.load_digits ([n_class, return_X_y])	Load and return the digits dataset (classification).
datasets.load_files (container_path[, …])	Load text files with categories as subfolder names.
datasets.load_iris ([return_X_y])	Load and return the iris dataset (classification).
datasets.load_linnerud ([return_X_y])	Load and return the linnerud dataset (multivariate regression).
datasets.load_mlcomp (*args, **kwargs)	DEPRECATED: since the http://mlcomp.org/ website will shut down in March 2017, the load_mlcomp function was deprecated in version 0.19 and will be removed in 0.21.

Load Dataset

- The datasets that start with fetch operation often need to download the data from internet.
  - For example, datasets.fetch_kddcup99()
- The datasets that start with load operation are already in the sklearn package.
  - For example, datasets.load_iris([*return_X_y=False*])
  - The *return_X_y* parameter decide whether the return value is (data, target) or an object with data, target, and DESCR as attribute.

Regressions

Linear and Logistic

Linear Regression

- In this example, we would like to predict the house price of Boston using linear regression
- First, we load the data

```
boston = datasets.load_boston()
```

- Let’s take a look at the data we get

```
>>> type(boston)
<class 'sklearn.utils.Bunch'>
>>> boston.keys()
dict_keys(['data', 'target', 'feature_names', 'DESCR'])
>>> boston.feature_names
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'],
      dtype='<U7')
```

Linear Regression

- More details about the data and attributes

```
Boston House Prices dataset

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target


 :Attribute Information (in order):
    - CRIM     per capita crime rate by town
    - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
    - INDUS    proportion of non-retail business acres per town
    - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    - NOX      nitric oxides concentration (parts per 10 million)
    - RM       average number of rooms per dwelling
    - AGE      proportion of owner-occupied units built prior to 1940
    - DIS      weighted distances to five Boston employment centres
    - RAD      index of accessibility to radial highways
    - TAX      full-value property-tax rate per $10,000
    - PTRATIO  pupil-teacher ratio by town
    - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
    - LSTAT    % lower status of the population
    - MEDV     Median value of owner-occupied homes in $1000's
```

Linear Regression

- Now that's build the linear regression model with the data
- First, initialize the model

```
lr = linear_model.LinearRegression(
fit_intercept=True,
normalize=False,
copy_X=True,
n_jobs=1)
```

Linear Regression

- Now we train the model

`fit (X, y, sample_weight=None)` [source]

Fit linear model.

Parameters:	**X** : numpy array or sparse matrix of shape [n_samples,n_features] Training data **y** : numpy array of shape [n_samples, n_targets] Target values. Will be cast to X's dtype if necessary **sample_weight** : numpy array of shape [n_samples] Individual weights for each sample *New in version 0.17:* parameter *sample_weight* support to LinearRegression.
Returns:	**self** : returns an instance of self.

Linear Regression

- Each row in the training data (X) represents one data
- Each column in the training data represents an attribute
- Each row in the target value represents the target value correspond to one data
- Each column of the target value is the different type of target we want to predict

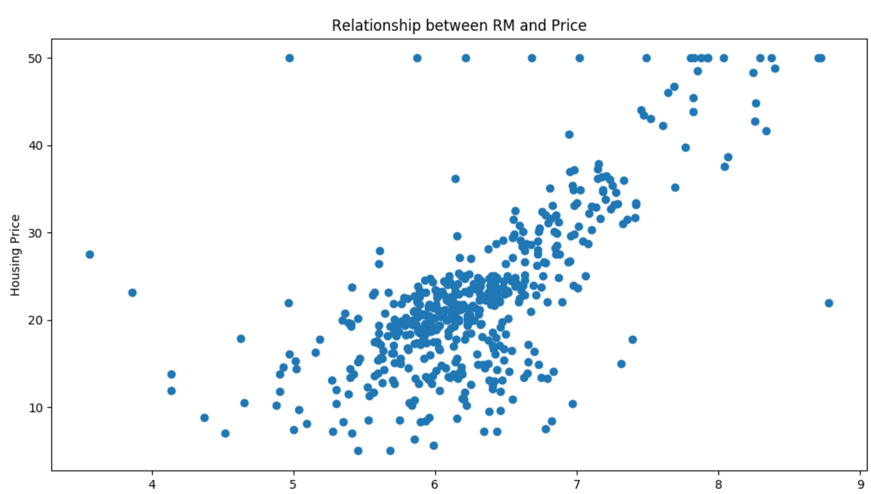
Linear Regression

```
>>> X = boston.data
>>> y = boston.target
>>> lr.fit(X,y)
LinearRegression(copy_X=True, fit_intercept=True,
n_jobs=1, normalize=False)
>>>
pd.DataFrame(list(zip(boston.feature_names,lr.coef_)),
columns=["Feature", "Correlation"])
```

Linear Regression

- From the correlation value we can find that the feature RM is highly correlated to the target
- Let’s plot the scatter plot of RM and Price

```
>>> from matplotlib import pyplot as plt
>>> plt.scatter(X[:,5], y)
>>> plt.xlabel("Average number of room per dwelling
(RM)")
>>> plt.ylabel("Housing Price")
>>> plt.title("Relationship between RM and Price")
>>> plt.show()
```



Linear Regression

- Now that's use the model to predict the house price. Here we predict the training data.

```
>>> predictY = lr.predict(X)
>>> predictY[0:5]
array([ 30.01,  25.03 ,  30.57 ,  28.61,  27.94])
>>> y[0:5]
array([ 24. ,  21.6,  34.7,  33.4,  36.2])
```

Linear Regression

- We can also plot the predict price and the real price

```
>>> plt.scatter(predictY, y)
>>> plt.xlabel("Predicted Price")
>>> plt.ylabel("Real Price")
>>> plt.title("Relation between the predicted and real
price")
>>> plt.show()
```

Cross validation

- To randomly divide the data, sklearn provides a function called train_test_split under the model_selection class

```
>>> X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X, y, test_size = 0.33)
>>> X_train.shape
(339, 13)
>>> X_test.shape
(167, 13)
>>> y_train.shape
(339,)
>>> y_test.shape
(167,)
```

Logistic Regression

- In this example, we would like to predict the class of the class if iris.
- Load the data with datasets.load_iris()
- There are 150 records and 4 attributes each.
- There are 3 different classes

Logistic Regression

- Initialize the logistic regression model with

```
linear_model.LogisticRegression(
penalty='l2',
solver='liblinear',
multi_class='ovr',
verbose=0,
n_jobs=1)
```

Logistic Regression

```
>>> logr = linear_model.LogisticRegression()
>>> Xtrain, Xtest, ytrain, ytest =
sklearn.model_selection.train_test_split(iris.data,
iris.target, test_size = 0.16)
>>> Xtrain.shape
(126, 4)
>>> logr.fit(Xtrain, ytrain)
>>> logr.score(Xtrain, ytrain)
0.96031746031746035
>>> logr.score(Xtest, ytest)
1.0
```

Logistic Regression

```
>>> logr2 =
linear_model.LogisticRegression(penalty='l1')
>>> logr2.fit(Xtrain, ytrain)
>>> logr2.score(Xtrain, ytrain)
0.96031746031746035
>>> logr2.score(Xtest, ytest)
0.95833333333333337
```

Logistic Regression

- When the problem is multi-class problem, there are generally 2 algorithms.
- One versus rest:
  - The algorithm compares every class with all the remaining classes, building a model for every class. If you have ten classes to guess, you have ten models.
- One versus one:
  - The algorithm compares every class against every - individual remaining class, building a number of models equivalent to n * (n-1) / 2, where n is the number of classes.

Logistic Regression

- We can modify the multi-class strategy using the “multi_class” parameter when initialize the model

```
>>> logr3 =
linear_model.LogisticRegression(multi_class='multinomia
l') # the default value is “ovr”
>>> logr3.fit(Xtrain, ytrain)
ValueError: Solver liblinear does not support a
multinomial backend.
```

Logistic Regression

```
>>> logr3 =
linear_model.LogisticRegression(multi_class='multinomia
l', solver = 'newton-cg')
>>> logr3.fit(Xtrain, ytrain)
>>> logr3.score(Xtrain, ytrain)
0.96031746031746035
>>> logr3.score(Xtest, ytest)
0.95833333333333337
```

Logistic Regression

- We can show the predicted probability that the sample belong to different classes

```
>>> logr3.predict_proba(Xtest[0].reshape(1, -1))
array([[  5.55e-05,   1.17e-01,   8.83e-01]])
```

Classification

SVM, Decision Tree, Neural Network

SVM

- In this example, we use SVM to classify the hand written digits.
- Load the dataset using datasets.load_digits()
- There are 1797 data and 64 attributes each
- The SVM of sklearn is based on libsvm

```
>>> digit = datasets.load_digits()
>>> digit.data.shape
(1797, 64)
>>> np.unique(digit.target)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

SVM

- Initialize the svm model with

```
sklearn.svm.SVC(
C=1.0,
kernel=’rbf’,
degree=3,
gamma=’auto’,
decision_function_shape=’ovr’
)
```

SVM

```
>>> svc_model = sklearn.svm.SVC(gamma=0.001, C=100.,
kernel='linear')
>>> svc_model.fit(Xtrain, ytrain)
>>> svc_model.score(Xtrain, ytrain)
1.0
>>> svc_model.score(Xtest, ytest)
0.97999999999999998
```

SVM

- In the previous example we set the C, gamma, and the kernel type
- However, these parameter greatly affect the performance of the SVM
- Unfortunately there is not any formula to find these values
- The grid search use brute force search to evaluate every possible combination of C, gamma and kernel type

SVM

- sklearn provides grid search cross validation function to help determine the parameter

```
>>> from sklearn.grid_search import GridSearchCV
>>> parameter_candidates = [
{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
'kernel': ['rbf']},]
>>> clf = GridSearchCV(estimator=svm.SVC(),
param_grid=parameter_candidates, n_jobs=-1)
>>> clf.fit(X_train, y_train)
```

SVM

```
>>> print('Best score for training data:',
clf.best_score_)
Best score for training data: 0.9844097995545658
>>> print('Best `C`:',clf.best_estimator_.C)
Best `C`: 10
>>> print('Best kernel:',clf.best_estimator_.kernel)
Best kernel: rbf
>>> print('Best `gamma`:',clf.best_estimator_.gamma)
Best `gamma`: 0.001
```

Dimensionality reduction

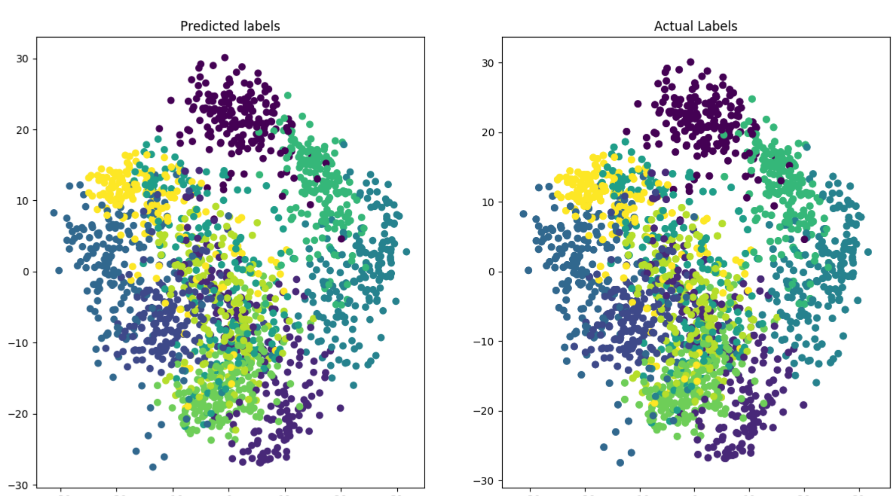
- Let’s visualize the result of SVM
- Frist we have to reduce the dimension of the data such that we can locate it on 2-D plane
- sklearn.manifold.Isomap
- sklearn.decomposition.PCA

Dimensionality reduction

- PCA

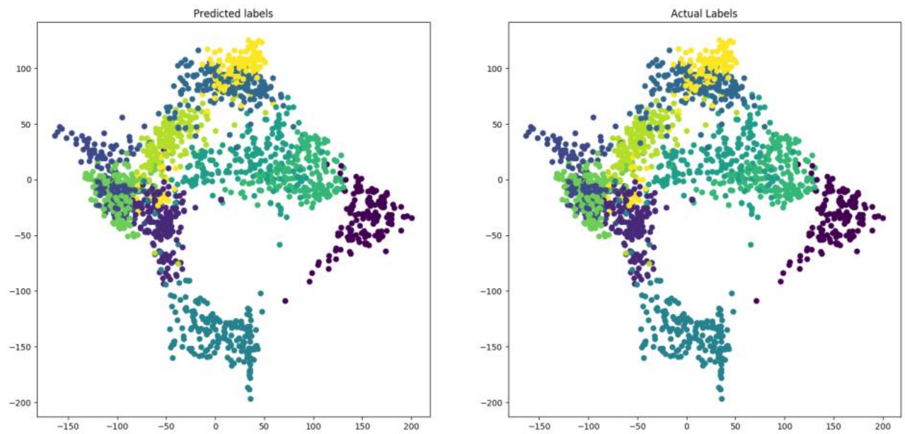
```
>>> from sklearn.decomposition import PCA
>>> PCA
<class 'sklearn.decomposition.pca.PCA'>
>>> pca = PCA(n_components = 2)
>>> pca.fit(digit.data)
>>> pdata = pca.transform(digit.data)
>>> pdata.shape
(1797, 2)
```

Plot



Isomap

```
>>> from sklearn.manifold import Isomap
>>> isoData = Isomap().fit_transform(digit.data)
```



Decision Tree

```
sklearn.tree.DecisionTreeClassifier(
criterion='gini',
splitter='best',
max_depth=None,
min_samples_split=2,
max_features=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0)
```

Decision Tree

```
>>> Xtrain, Xtest, ytrain, ytest =
sklearn.model_selection.train_test_split(iris.data,
iris.target, test_size = 0.17)
>>> dct = sklearn.tree.DecisionTreeClassifier()
>>> dct.fit(Xtrain, ytrain)
>>> dct.score(Xtest, ytest)
0.92307692307692313
>>> dct.score(Xtrain, ytrain)
1.0
```

Decision Tree

- You can export the tree with sklearn.tree.export_graphviz()
- The output file will be .dot format
- Graphviz is an open source graph visualization software.
- Represent structural information as diagrams of abstracted graphs and networks

Decision Tree

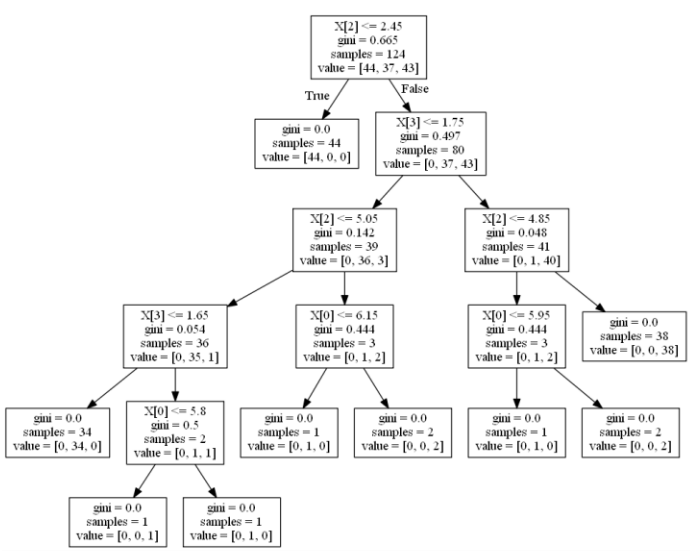
- Generate the tree structure file using

```
>>> sklearn.tree.export_graphviz(dct,
out_file="tree.dot")
```

- Use the graphviz tool to generate the picture

```
> dot.exe -Tpng tree.dot -o tree.png
```

Decision Tree



Neural Network

- sklearn.neural_network.MLPClassifier
- MLP stand for Multi-Layer Perceptron
- MLP is sensitive to feature scaling, so it is highly recommended to scale the data first.
- Either map the data to [0,1] or [-1, +1], or standardize it to have mean 0 and variance 1
- sklearn provides an easy way for scaling the data

Data scaling

- StandardScaler(copy=True, with_mean=True, with_std=True)

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler()
>>> scaler.fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
```

Neural Network

```
sklearn.neural_network.MLPClassifier(
hidden_layer_sizes=(100, ),
activation='relu',
solver='adam',
alpha=0.0001,
batch_size='auto',
learning_rate='constant',
learning_rate_init=0.001,
max_iter=200,
shuffle=True,
momentum=0.9,)
```

Neural Network

```
>>> Xtrain, Xtest, ytrain, ytest =
sklearn.model_selection.train_test_split(iris.data,
iris.target, test_size = 0.165)
>>> scalar.fit(Xtrain)
>>> Xtrain2 = scalar.transform(Xtrain)
>>> Xtest2 = scalar.transform(Xtest)
>>> mlp = MLPClassifier(hidden_layer_sizes = (15,))
>>> mlp.fit(Xtrain, ytrain)
>>> mlp.score(Xtrain, ytrain)
0.83999999999999997
>>> mlp.score(Xtest, ytest)
0.80000000000000004
```

Neural Network

```
>>> mlp.fit(Xtrain2, ytrain)
>>> mlp.score(Xtrain2, ytrain)
0.872
>>> mlp.score(Xtest2, ytest)
0.88
```

Clustering

KMeans

KMeans

- In this example, we are going to cluster the hand written digit from the digit dataset
- Since we know that there are only 10 digits, we know there should be 10 clusters

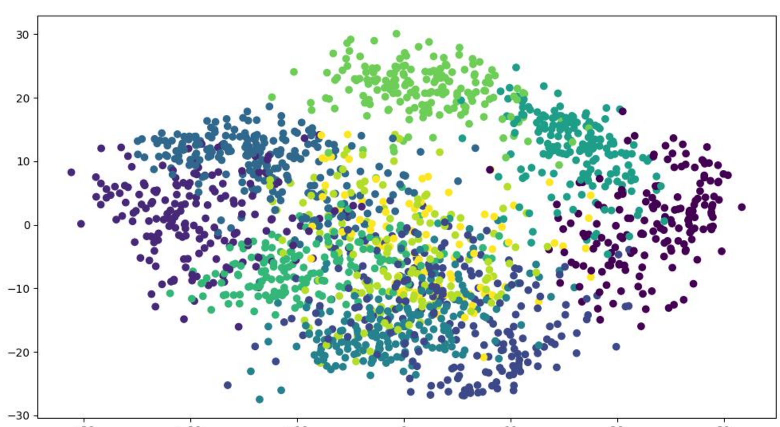
KMeans

```
sklearn.cluster.KMeans(
n_clusters=8,
init='k-means++',
n_init=10,
max_iter=300,
)
```

KMeans

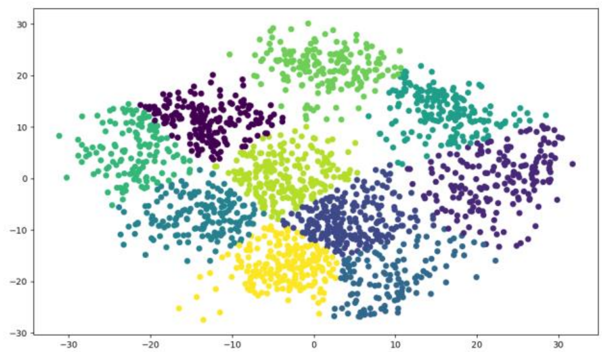
```
>>> data = digit.data
>>> reduced_data = PCA(n_components=2).fit_transform(data)
>>> kmeans = KMeans(init='k-means++', n_clusters=10)
>>> kmeans.fit(data)
>>> result = kmeans.predict(data)
>>> plt.scatter(reduced_data[:,0], reduced_data[:,1], c =
result)
>>> plt.show()
```

KMeans



KMeans

```
>>> kmeans.fit(reduced_data)
>>> result = kmeans.predict(reduced_data)
>>> plt.scatter(reduced_data[:,0], reduced_data[:,1], c
= result)
>>> plt.show()
```



Reference

- http://blog.csdn.net/puqutogether/article/details/42971617
- https://machine-learning-python.kspax.io/Introduction/intro.html
- http://dataaspirant.com/2017/02/01/decision-tree-algorithm-python-with-scikit-learn/
- http://scikit-learn.org/stable/

HW2

Spam letter classification

Spam letter classification

- Use the sklearn library to determine whether a letter is spam letter or not.
- The dataset is downloaded from https://archive.ics.uci.edu/ml/datasets/spambase
- 57 attributes, the last attribute marks the class of the data.

Spam letter classification

- In this homework, you will have to do the classification in 4 ways
  - Regression
  - Decision Tree
  - SVM
  - Neural Network

Format and rule

- You should hand-in only one file classification.py alone with one report.pdf zipped into one file
  - We will execute the script with the following command
  - `python classification.py [R, D, S, N] train.csv test.csv`
  - The R, D, S, N determines what method you are going to use to classify
  - Only one method will be specified each time
  - The train.csv contains all attributes include the class information
  - The test.csv does not contain the class information
  - An example of train.csv and test.csv is provided in the homework file.
  - You should generate one “predict.csv” every time we execute the script

Format

- The predict.csv contains the same number of rows with test.csv
- Each row has only one number 1 or 0, which is the predicted class of the test
- You should describe the design of your model in the report.
- You should also compare the accuracy of different method and explain the reason
- The training data and the testing data will be split randomly from the data when judging the score.
- Do remember to cross validate your model

Scoring

- There are 4 method, each worth 20 points (80% total)
  - 15 points for correct implementation
  - 5 points for the model accuracy
    - The top 30 students get 5 points
    - Passing the baseline gets 4 points
- The report worth 20 points
- Do not pre-train your model with the data
- The late submission penalty is 15 points per day
- We accept late submission for at most 2 days.