Machine Learning H/M Coursework

Group M20

Ching-Han Cheng	2413917C
Yidi Cao	2416218C
Ying Xu	2395159X
Zhuohao Wang	2374389W

Introduction

An efficient tool *sklearn* is adopted to conduct regression and classification analysis of data in this coursework.
Several packages from *sklearn* are used for four models. In the regression analysis of CPU data, El asticNet and Bayesian automatic relevance determination regression models are implemented, with some parameters updated to find the optimal solutions. In the classification analysis of cells, Naive Bayesian and SVM (Support Vector Machine) classification models are adopted. Finally, we chose better models according to the scores from *Kaggle*.

Dataset

There are 168 CPU training sets in the data of regression analysis, which contains the six character istics of CPU. Our task is to predict the performance scores for 41 new CPUs.
In the data of classification analysis, there are 600 items with 112 features for each item. The data are expected to be grouped into two categories, "1" for the epithelial region and 2 for the strom al region. Our task is to predict categories of a new set of data.

Sklearn

Sklearn is a very useful tool for machine learning in Python, helping us analyse data from various contexts. This tool contains many packages for different purposes, like preprocessing data, perfor ming regression or classification analysis on data. There are several general functions used in this coursework.

Split arrays or matrices into the random training and testing subsets.

sklearn.model_selection.**train_test_split**(**arrays*, ***options*)[source]

K-Folds cross-validator

*class*sklearn.model_selection.**KFold**(*n_splits='warn',shuffle=False,random_state=None*)[source]

Pipeline of transforms with a final estimator.

*class*sklearn.pipeline.**Pipeline**(*steps,memory=None*)[source]¶

C-Support Vector Classification

*class*sklearn.svm.**SVC**(*C=1.0,kernel='rbf',degree=3,gamma='auto_deprecated',coef0=0.0,shrinking=True,probability=False,tol=0.001,cache_size=200,class_weight=None,verbose=False,max_iter=-1,decision_function_shape='ovr',random_state=None*)[source]¶

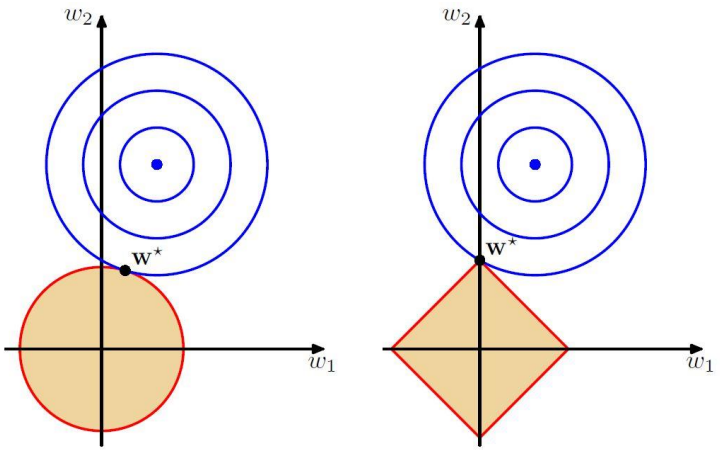
Exhaustive search over specified parameter values for an estimator.

*class*sklearn.grid_search.**GridSearchCV**(*estimator,param_grid,scoring=None,fit_params=None,n_jobs=1,iid=True,refit=True,cv=None,verbose=0,pre_dispatch='2*n_jobs',error_score='raise'*)[source]¶

Regression

ElasticNet regression

ElasticNet is a linear regression with L1 and L2 priors as regularizer. Regularization is a common method used in linear regression. This method could prevent overfitting of model parameters, and some constraints are added to parameters in the process of parameter optimization. L1 regular prio r is often used in lasso regression method, L2 regular prior is often used in ridge regression. Theref ore, Elastic regression is a combination of Lasso regression and Ridge regression.
The solution space of the L2 norm is a circle, and the solution space of the l1 norm is a quadrilater al. The solution spaces of the L1 and L2 norm are shown in the figure below.

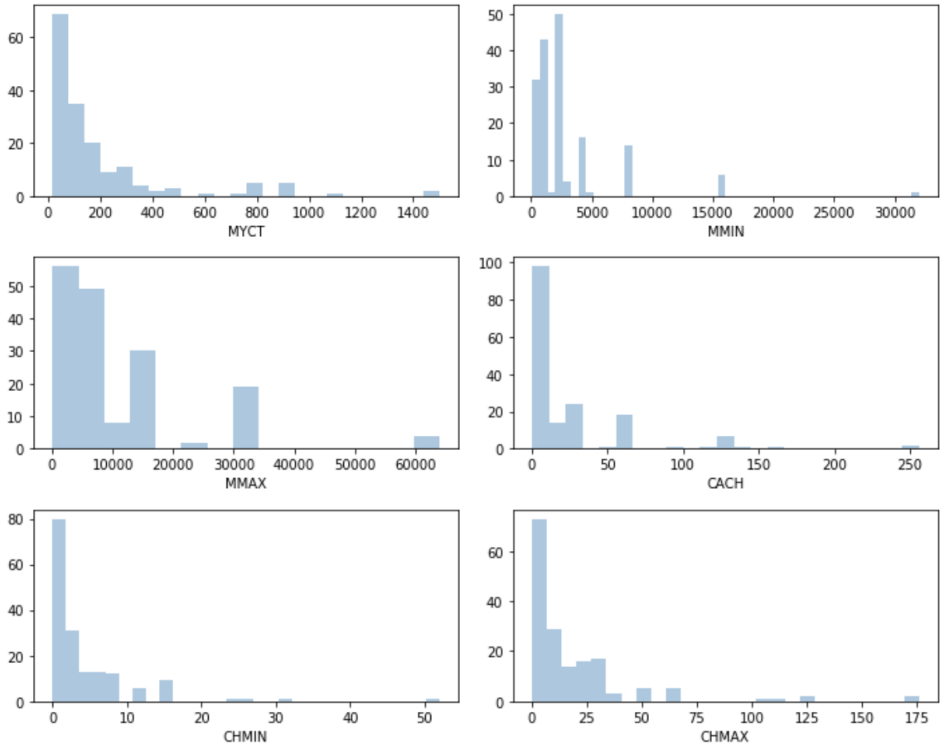


In the elastic regression, the L1 and L2 regular terms are weighed, and the objective function is:

$$\min_{w} \frac{1}{2n_{samples}} ||Xw - y||_2^2 + \alpha\rho||w||_1 + \frac{\alpha(1-\rho)}{2}||w||_2^2$$

Experiment setup

The distribution of these six features is shown graphically using seaborn.



There is a package of elastic regression in sklearn, named `sklearn.linear_model.ElasticNet`.

*class*sklearn.linear_model.**ElasticNet**(*alpha=1.0,l1_ratio=0.5,fit_intercept=True,normalize=False,precompute=False,max_iter=1000,copy_X=True,tol=0.0001,warm_start=False,positive=False,random_state=None,selection='cyclic'*)[source]¶

Firstly, train_test_split was used to segment data. Taking 20% of the data as a test set. Because aft er our calculation, the result is the best when test_size is equal to 0.2. Secondly, the Pipeline functi on could scale and construct feature polynomials. Thirdly, in the ElasticNet function, set the value of alpha, but we don't know what the optimal alpha value is. We use the for loop to find the optim al value. At this time, the value of the objective function is the smallest. It is calculated that when alpha is equal to 0.313, the value of Model MAE is the smallest. The fit of the model is evaluated by comparing the difference between y_val and y_pred. Some other evaluation methods are also a dopted, and the results are shown in the figure below. Finally, fit this model with the whole data a nd X_test is used to generate the corresponding CSV file.

```
Model MAE: 16.660352546671447
Explained Variance Score: 0.9125139044358148
Mean Squared Error: 600.5693455656001
Median_Absolute_Error: 12.03919303562298
r2_score: 0.912054563218904
```

Bayesian Automatic Relevance Determination (ARD) regression

Model behind the regression algorithm

ARDRegression is a probabilistic model of the regression problem:

$$p(y|X, \omega, \alpha) = N(y|X\omega, \alpha)$$

The regularization is included in the estimation process. The regularization is to find a maximum a posterior estimation under a Gaussian prior over the parameters ω with precision λ^{-1}. λ and α a re both random variables to be estimated from data. The parameters ω, α and λ are estimated jo intly during the fit of the model. The distribution of ω is an axis-parallel, elliptical Gaussian distri bution:

$$p(\omega|\lambda) = N(\omega|0, A^{-1})$$

with $\text{diag}(A) = \lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_p\}$.

Each coordinate of ω_i has its own standard deviation λ i. The prior over all λ i is chosen to be t he same γ distribution given by hyper parameters λ 1 and λ 2.

It is very similar to Bayesian Ridge Regression.

Experiment setup

step1 read data

First, read train/test data from files, which is the same as the process in Elastic Net Regres sion.

Step2 split data

Second, split all the data into four datasets using “train_test_split”, including train data x, train targets y, test data x, test targets y, of which x represents for six features and y stands for performance scores. The size of test/train dataset can be set as parameters in the method train_test_split(). After several tests and analysis, we found that when the test dataset takes 10% of the whole dataset, the result turns to be the best.

Then, pipeline () is used to do further data processing. The method PolynomialFeatures() is used to generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree, and here we set the degree as 2 and only interaction features are produced. StandardScaler() is utilized to standardize features by removing the mean and scaling to unit variance. Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the transform method. The method fit_transform() of Pipeline is used to fit all the transforms and transform the train data x, then transformed data is processed with the final estimator. Then, the method pipeline.transform() applies the transforms and transform test data x with the final estimator.

Step 3 fit the model with an ARD prior

This step is to fit the regression model with train data x and train targets y using the method linear_model.ARDRegression().fit(), and then generate the prediction value of test data x as y_pred. After that, calculate the values of several measures of the performance of the model. The results are as follow:

```
Model MAE: 13.11400519858643
Explained Variance Score: 0.9456972843819389
Mean Squared Error: 289.26243493178566
Median_Absolute_Error: 8.529083009323386
r2_score: 0.9423240930912652
```

Step 4 make prediction

Refit the model using the whole given dataset. The whole dataset is first transformed with pipeline() and then the model is fitted with linear_model.ARDRegression().fit(). Finally, predict the test targets using ARD regression model and store the results into a CSV file.

Measure the performance

several indicators are used to measure the performance of the model.

Mean absolute error is a measure of difference between two continuous variables. The smaller the number, the better the result.

Explained variance score is to measure the variance between the estimated target output and the correct target output. The best possible score is 1.0, lower values are worse.

Mean Square Error (MSE) is the average of the square of the errors. The larger the number, the lar ger the error.
r2 score is the total variance explained by model. It ranges from 0 to 100%. If it is 100%, then two variables are perfectly correlated. A low value would show a low level of correlation, meaning a re gression model that is not valid.

Compare the performance of the two algorithms

According to the results above, the performances of the two models are as follow:

	Model M AE	Explained Variance Score	Mean Squared E rror	Median Absolute Error	r2_sco re
ElasticNet	16.6603	0.9125	600.5693	12.0391	0.912
Bayesian A RD	13.114	0.9456	289.2624	8.529	0.9423

For every measure, Bayesian ARD Regression has a better performance than Elastic Net Regressio n. As a result, we can make a conclusion that Bayesian ARD Regression is more suitable for our s cenario.

Classification

Read the data

```
In [1000]: df = pd.read_csv('X_train.csv')
           y = pd.read_csv('y_train.csv')['EpiOrStroma']
           df_test = pd.read_csv('X_test.csv')
           df.head()
```

Out[1000]:

	Mean.Layer.1	Mean.Layer.2	Mean.Layer.3	Standard.deviation.Layer.1	Standard.deviation.Layer.2	Standard.deviation.Layer.3	Skewness.Layer.1
0	125.858498	30.964032	164.498814	43.494697	31.765953	32.182953	0.140945
1	96.203484	25.650567	142.116917	53.263910	40.001324	45.085541	0.502491
2	119.004390	34.141564	162.757202	53.132611	50.290571	38.863597	0.552590
3	127.335236	32.264460	170.291857	45.569112	29.347426	32.034223	-0.201071
4	179.483106	66.308868	185.935024	41.050604	51.946844	37.133099	-0.273187

Exploratory Analysis

Check the distribution of the dataset.

```
df.describe()
```

	Mean.Layer.1	Mean.Layer.2	Mean.Layer.3	Standard.deviation.Layer.1	Standard.deviation.Layer.2	Standard.deviation.Layer.3	Skewness.Layer.1	S
count	600.000000	600.000000	600.000000	600.000000	600.000000	600.000000	600.000000	
mean	182.301111	118.986652	202.973900	38.288443	44.890472	32.093265	-0.791233	
std	36.665231	46.897067	22.244349	10.613177	10.078749	8.007605	0.726370	
min	67.388614	25.650567	123.014851	4.077094	11.614268	3.902983	-3.710839	
25%	150.899714	82.373131	186.165918	31.617634	37.890627	27.318182	-1.265568	
50%	191.652576	115.394283	204.051131	39.364728	45.226574	32.573667	-0.728627	
75%	212.704532	156.620265	220.319194	45.320071	51.513966	37.077982	-0.227479	
max	251.609800	242.569873	252.339383	70.625319	74.427457	56.322197	0.583831	

8 rows × 112 columns

There are 112 variables in the dataset.
Check if there is Nan in the dataset.

```
In [921]: df.isna().head()
          print("*****In the training set*****")
          print(df.isna().sum())
          print("\n")
          print("*****In the testing set*****")
          print(df_test.isna().sum())
```

```
*****In the training set*****
Mean.Layer.1                    0
Mean.Layer.2                    0
Mean.Layer.3                    0
Standard.deviation.Layer.1      0
Standard.deviation.Layer.2      0
Standard.deviation.Layer.3      0
Skewness.Layer.1                0
Skewness.Layer.2                0
Skewness.Layer.3                0
Ratio.Layer.1                   0
Ratio.Layer.2                   0
Ratio.Layer.3                   0
```

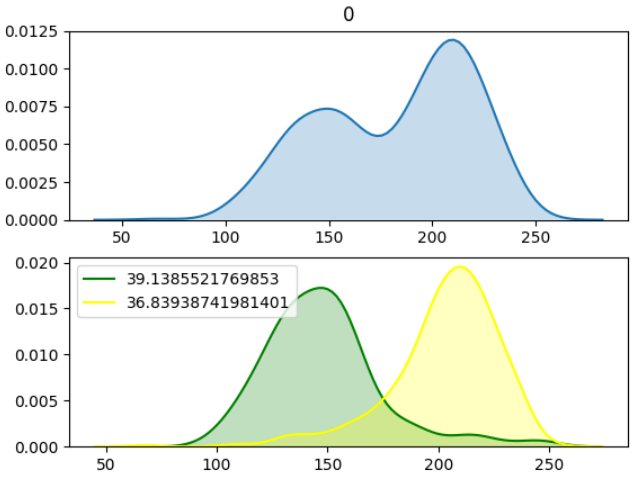
There is not any Nan in the dataset.
Check the distribution of each variable.

Subset of the features

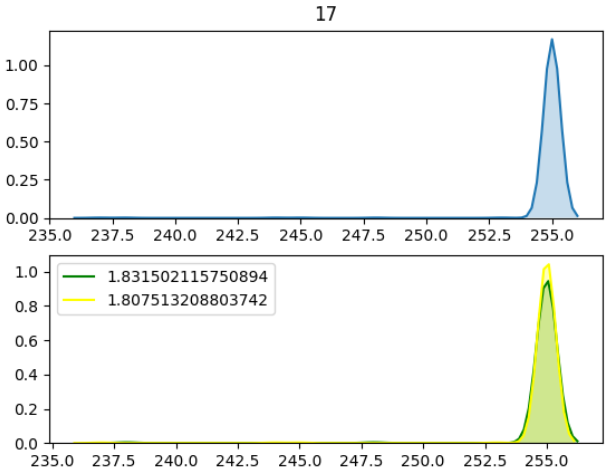
Considering there is a huge number of features, it is likely that some features are useful for the cla ssification. Therefore, it is likely to be better if we remove some useless

Step 1: Observe the distribution of data

The idea is that we could paint two distributions of two kinds of cells for each feature, if the two d istributions are quite similar, this feature will be regarded as noise. The reason is that this feature h as no contribution for classification of these two cells.



This graph shows the overall distribution and two distributions for each kind of cells separately. T his feature ‘0’ is expected to be useful since it shows different trends of two cells.



As for feature ‘17’, the trends of two cells are really similar, so this feature is considered to be useless and would be removed.

As a sequence, a list of useless features is obtained, which is [17,40,41,65,66,67,68,79,80,81,83,85, 87]（following the order of feature in file）

The accuracy of models will be improved after removing these features by approximately 0.2%.

Cross validation

Split dataset with 90% training data and 10% testing data.

In [953]:
```
# split the data
X, y = df.values, y.values
X_train, X_val, y_train, y_val = train_test_split(df, y, test_size=0.1, random_state=1)
print('train shape:', X_train.shape, 'validation shape:', X_val.shape)
print(X_train.shape)
print(X_val.shape)
print(y_train.shape)
print(y_val.shape)
```

```
train shape: (540, 112) validation shape: (60, 112)
(540, 112)
(60, 112)
(540,)
(60,)
```

Building the Pipeline

Step 1: Normalize data

Step 2: Apply classifier method

```
In [23]: pipeline = Pipeline([
             ('normalizer', StandardScaler()), #Step1 - normalize data
             ('clf', SVC()) #step2 - classifier
         ])
         pipeline.steps
```

```
Out[23]: [('normalizer', StandardScaler(copy=True, with_mean=True, with_std=True)),
          ('clf', SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
             kernel='rbf', max_iter=-1, probability=False, random_state=None,
             shrinking=True, tol=0.001, verbose=False))]
```

```
pipeline = Pipeline([
    ('normalizer', StandardScaler()), #Step1 - normalize data
    ('clf', GaussianNB()) #step2 - classifier
])
pipeline.steps
```

```
[('normalizer', StandardScaler(copy=True, with_mean=True, with_std=True)),
 ('clf', GaussianNB(priors=None, var_smoothing=1e-09))]
```

We have applied several classifier methods
SVM
KNN
Decision Tree
Random forest
Gaussian Naive Bayes

```
clfs = []
clfs.append(LogisticRegression())
clfs.append(SVC())
clfs.append(KNeighborsClassifier(n_neighbors=2))
clfs.append(DecisionTreeClassifier())
clfs.append(RandomForestClassifier())
clfs.append(GaussianNB(priors= [0.5, 0.5]))
```

Then compare the performance of each model.

```
from sklearn.model_selection import cross_validate
scores = cross_validate(pipeline, X_train, y_train)
scores
```

The performance of each model: (mean test_score)
LogisticRegression: 0.90930881497257
SVC: 0.9074465032642755
KNeighborsClassifier: 0.8722713206740491
DecisionTreeClassifier: 0.8944316982805826
RandomForestClassifier: 0.9093498541907442
GaussianNB: 0.8815101174819068

These five models are provided for training models, then the predictions of each model are submitted on Kaggle as the feedback.
We first thought the performance of Random Forest Classifier and Logistic Regression might be the best, however, the result of SVC and Gaussian Naive Bayes shows better performance on Kaggle. Therefore, we suggest the reason could be the dataset we got is too small compares to the entire dataset.
At last, we chose SVM and Gaussian Naive Bayes as our classifier models.

The performance of SVM is 0.91.

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
---------------------------------
fit_time  mean  0.01063863436381022
fit_time  std  0.0009406052873113953
score_time  mean  0.003989219665527344
score_time  std  0.000814004069894147
test_score  mean  0.9074465032642755
test_score  std  0.013544200092304138
train_score  mean  0.9675924354125961
train_score  std  0.0026197196769513734
```

The performance of Gaussian Naive Bayes is 0.88.

```
GaussianNB(priors=[0.5, 0.5], var_smoothing=1e-09)
---------------------------------
fit_time  mean  0.002972682317097982
fit_time  std  0.0008023658947018878
score_time  mean  0.0003439585367838542
score_time  std  0.00048643082761373166
test_score  mean  0.8815101174819068
test_score  std  0.015745244607408844
train_score  mean  0.8851923082968461
train_score  std  0.003207133676996521
```

SVM (Support vector machine)

The main idea of SVM is to put the characteristics of data to a high dimension and use hyper plane to split the data points of different categories. The support vector is composed of data points of dif ferent classes, which determines the final division of the hyperplane.

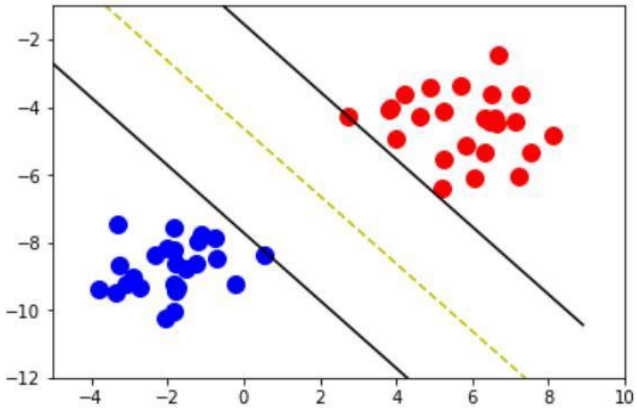


Figure Support vector machine (from https://www.google.co.uk/search?q=support+vector+machine&source=lnms&tbm=isch&sa=X&ved=0ahUKEwjCyarh0fHeAhULBcAKHcUDC6QQ_AUIDigB&biw=1225&bih=675#imgrc=SHNbgL3fuF0XcM:)

In Figure 1, the yellow dotted line is the hyperplane (If the dimension of data is n, the dimension o f the hyperplane is n-1). We try to make the distance between the two support vectors (two black s olid lines) of two classes (blue and red dots), are the largest.

The expression of yellow dotted line is

$$ax + by + c = 0$$

Therefore, we can get the expression of hyper plane in high dimensions

$$\omega^T x + b = 0$$

We also get the expression of division the two different classes

$$\omega^T x + b > 0$$
$$\omega^T x + b < 0$$

The expression of the distance is

$$distance = \frac{\omega^T x + b}{||\omega||}$$

We can get the values of ω and b by calculating the Maximum.

In practice, the data would not always be divided perfectly. It is likely that some data is divided in the wrong class. We expect the model to have better generalization ability, we hope that the middl e transition zone ρ is as large as possible, so we will selectively ignore some bad points. ξ is int roduced as the slack variable and the model becomes

$$\min_{\omega,b,\xi} \frac{||\omega||^2}{2} + C\sum_{i=1}^{n} \xi_i$$

The larger C, the more we tend to have no slack variables, that is, the model will divide every data point as correctly as possible. Conversely, the smaller the C, the stronger the generalization ability of the model.

The method of high dimensional transformation is called kernel function. In general, there are 4 di fferent kernel functions, linear, Polynomial, RBF radial basis and sigmoid.

With powerful package *sklearn.svm.SVC*, we can implement this simply. With another powerful p ackage *sklearn.grid_search.GridSearchCV*, we can adjust the parameters. Finally, we can get the b est prediction with kernel = 'sigmoid' and C = '0.18', which results in the accuracy of 0.91 667.

Gaussian Naive Bayes

Model of Gaussian Naive Bayes:

$$x = [x_1, x_2, \ldots, x_{121}]$$

$$k = 1,2$$

$$p(t_{new} = k|X, t, x_{new}) = \frac{p(x_{new}|t_{new} = k, X, t)p(t_{new} = k)}{\sum_j p(x_{new}|t_{new} = j, X, t)p(t_{new} = j)}$$

Binomial likelihood:

$$p(x_{new}|t_{new} = k, X, t)$$

prior:

No prior preference: $p(t_{new} = 1) = p(t_{new} = 2) = 0.5$

Step1. Select the GaussianNB model

Step2. Set the prior as [0.5, 0.5]

```
GridSearchCV(cv='warn', error_score='raise-deprecating',
       estimator=Pipeline(memory=None,
     steps=[('normalizer', StandardScaler(copy=True, with_mean=True, with_std=True)), ('clf', GaussianNB(priors=[0.5, 0.5], var
_smoothing=1e-09))]),
       fit_params=None, iid='warn', n_jobs=None, param_grid={},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

Step3. Fit the model with cv_grid.fit

90% of training data are provided.

Step4. Apply the grid_predict for checking the accuracy of our model after cross val idation

```
Accuracy of the best classifier after CV is 90.000%
```

The result of Gaussian Naive Bayes is 90%.

Step5. Fit the model with cv_grid.fit

100% of data are provided

Step6. Use the model for making the prediction with cv_grid.predict.

```
Predictions [2 2 1 ... 1 1 2]
 Data shape (1596,)
```

Literature Review

Paper title: Auto-Encoding Variational Bayes

The problem is how to perform efficient inference and learning in directed probabilistic models, w hen continuous latent variables have intractable posterior distributions and the datasets are large. T he state- of- the- art before the paper is that, variational Bayesian approach can optimize an approx imation of the intractable posterior; however, its requirement of analytical solutions of expectation s is also intractable.

The novel contribution that the authors made includes two aspects. First, they derive a Stochastic Gradient Variational Bayes (SGVB) estimator of the variational lower bound for approximate post erior inference. This estimator can be optimized straightforwardly using standard stochastic gradie nt ascent techniques. Second, they propose the Auto-Encoding VB algorithm with SGVB estimator to make inference and learning for i.i.d. datasets with continuous latent variables per datapoint. In t he experiment, the authors adopt the MNIST and Frey Face datasets to train generative models of i mages and use the variational lower bound and the estimated marginal likelihood as measures to c ompare the performances of AEVB, the wake-up algorithm and Monte Carlo EM (MCEM). They first compared the AEVB and wake-up algorithm in terms of their performances on optimizing the lower bound for different dimensionality of latent space. The results showed that AEVB has a high er converge speed and performed better. Then, they compared the AEVB, Wake-Sleep method and MCEM with a Hybrid Monte Carlo (HMC) sampler for different sizes of training datasets, using t he estimated marginal likelihood as a measure. The results showed that MCEM is unable to be app lied for the whole MNIST dataset and AEVB has the best converge speed.

Paper title: Distributed Representations of Words and Phrases and their C ompositionality

The problem is that many phrases have a meaning that is not a simple composition of the meaning s of its individual words. Therefore, the author improves the existing model to obtain the relations hip between sentences and words. The state-of-the-art before the paper are the Skip-gram model a nd Hierarchical Softmax. The Skip-gram model can output the words in the context of the training set based on the input of a word in the training set. All of these words are then coded to calculate t he probability of the Cth word output. Hierarchical Softmax binary trees to find the relative probab ilities of its child nodes. Therefore, it is just needed evaluate about $\log_2(W)$ nodes.

This paper proposes many new solutions. Firstly, to improve the Hierarchical Softmax method, the author used Huffman tree to assign short codes for frequent words, which could speed up the traini ng. Additionally, the authors proved that the word and phrase representations learned by the Skip-gram model exhibit a linear structure. Secondly, Negative Sampling, an alternative to Hierarchical Softmax method. In the process of operation, this solution does not need the distribution of noise b ut use logistic regression to distinguish the target word w_0 from the noise distribution $P_n(w)$. In addi tion, each data sample has k negative samplings. Thirdly, subsampling of Frequent Words, because there are some frequency words do not provide information value. This approach can compensate f or rare words and frequent imbalances: each word w_i in the training set is discarded with probabilit

y computed by the formula. Finally, a simple data-driven approach is used to calculate phrases bas ed on unigram and bigram, which can reduce the memory intensive. In the evaluation of the above methods, the authors used the analogical reasoning task introduced by Mikolov et.al.. This task has two broad categories: the syntactic analogies and the semantic analogies. In addition, the authors d iscarded words that occurred less than 5 times from vocabulary in the training data. Finally, accord ing to the results of the table shows negative sampling on the analogical reasoning task is superior to the Hierarchical Softmax, and even than noise estimation has better performance. Sub-sampling of frequent words can improve training speed several times, making word representation more acc urate. Since there are different optimal parameter configurations for different issues, the authors us e Skim-gram model to adopt different hyperparameter values when training phrases and then incre ased the number of training data, making the accuracy rate reach 72%.

Paper title: Computational analysis of cell-to-cell heteroge neity in single-cell RNA-sequencing data reveals hidden subpopulations of cells

Scientists become increasingly interested in subpopulations of cells. The previous approaches can only examine the expression of a small number of genes. Therefore, the research of co-expression factors and the identification of subpopulations are not clear.

As for single-cell measurements, RNA-FiSH has been used to show the insights of transcriptions. Protocols have been developed to improve the identification of factors, by amplifying small numbe r of mRNA. With next-generation sequencing, some approaches have been used to model early em bryogenesis in the mouse and to investigate bimodality in gene expression patterns of differentiatin g immune cell types. After this great step, scientists want to identify subpopulations. However, me thods for identifying subpopulations of cells and modeling are not developed.

The novel contribution of the article is to present and validate a computational approach that uses l atent variable models to account for such hidden factors. Authors demonstrate single-cell latent var iable model (scLVM) used for the identification of subpopulations of cells with experiment about t he differentiation of naive T cells into T helper 2 cells. Consequently, scLVM is an effective and r obust method to reconstruct higgen factors and achieve the identification of subpopulations.

There are two steps for scLVM algorithm. First, one or more covariance structures are inferred fro m genes that are annotated to hidden factors such as cell-cycle progression. Subsequently, these co variance structures can be used to account for the hidden factors as random effects in a mixed mod el, allowing the variance in expression for each gene to be decomposed into a technical, a biologic al and a separate component for each hidden factor.

Authors valite scLVM with mouse embryonic stem cells and T helper 2 (TH2) cell differentiation. Firstly, it is demonstrated that Cell cycle variation affects global gene expression. Then author use scLVM to find the confounding effects of cell cycle, which shows that scLVM can be used to robu stly recover and estimate the variance. Subsequently, the study of cell populations in differentiatin g TH2 cells are processed with our scLVM approach. After the examination of the credibility of th e scLVM model, two clear subpopulations of cells were identified. To demonstrate that scLCM ca n work in the case of more than one factor, it is applied to account for the effects of other factors, t he TH2 cells by simultaneously modeling the cell-cycle state and the TH2 differentiation.

Paper title: Generative Adversarial Nets

This paper resolves several problems and mainly the situations happen in semi-supervised or unsu pervised learning.
One of the problems is that people do not have enough data for training the model so it may easily cause the model overfitting. Also, the labeled data human generate is often limited since the cost o f creating these data are high. Finally, the most serious problem is when there are too many intract able probabilities need to be computed which may spend a huge amount of computation resources or time.
As the result, the authors aim to decrease the cost of human as providing training data and eliminat ing the calculations such as the Markov chain in the steps of training the model.

Most previous research works on generative models, and they tried to figure out the most suitable parameters for the probability distribution function. Moreover, the way to maxima the performance of gradient descent. For instance, authors think the deep Boltzmann machine [1] could be the most successful approach for solving complicated optimisation problems.
However, the model has typically complicated likelihood functions, so it will also need to address the situations calculating numerous approximations of gradient descent.

The authors provide the novel approach about 'Generative Adversarial Network' as the solutio n, the idea is to generate mainly three functions namely NCE, GAN and VAE. At first, the is gener ated as the noise and then G(z; θ g) is provided for generating new datasets, at last, D(x; θ d) is cre ated for detecting the fake datasets among the datasets.
The solution aims to create a condition that two main functions are conflict, so when the performa nce of both functions getting better, the model then can automatically generate training data and d o the training by itself. The process of the methodology is to find the minimum of probabilities tha t GAN is found as generating fake dataset and maxima the probabilities that D can recognize whic h dataset is the not the real one. As a result, the function is shown below.
This new approach reduces the cost of human while the model can be trained and improved autom atically by itself. Therefore, this can be widely provided in various fields such as visualising the sa mples from the model or generate some masterpiece painting.
In conclusion, this novel approach can not only reduce the cost of training model but also break th e restriction of calculation and limit amount of data.
Reference:
[1] Bengio, Y., Thibodeau-Laufer, E., and Yosinski, J. (2014a). Deep generative stochastic networ ks trainable by backprop. In ICML' 14.