EE559 Final Project

Data Set(s): Adult

Pin-Hsuan Lee, pinhsual@usc.edu

Date: 04/30/19

1. Abstract

In the project, we try to deal with the adult data set from UCI, which includes several information, such as age, marital status, native country, … etc. We are trying to deal with all of the information and to predict that whether a person's income is greater or less than $50K per year by the given information. In the process, we preprocess the data, apply sequential backward feature selection, implement cross-validation to parameter selection, etc. We take perceptron, SVM, and KNN as our classifiers, and get the best results which equal to 85.98% in training data and 85.32% in test data in SVM.

2. Introduction

2.1. Problem Statement and Goals

In this project, I choose to deal with the adult data sets. Because there are some problems in the data set, I need to try to preprocess the data, find the optimal parameters, and then apply some ways to enhance the performances of the classifiers when predicting the labels.

2.2. Literature Review

The data sets include the problem of missing data, imbalance data, and redundant features. There are many ways to deal with the missing data like removing the data points that have the missing features inside, filling the missing data with the mean or the mode of the feature, etc. When talking to addressing the imbalance data, many people would try to replicate the data which belongs to the minority labels by using SMOTE or ADASYN in Python. Besides, when there are too many features, we can use feature selection to

reduce the dimension of feature space. There are several approaches to deal with these problems in the data set.

3.Approach and Implementation

3.1. Preprocessing

First, we loaded the data as a dataframe in Python and assigned the column into “age”, “workclass”, ”fnlwgt”, ”education”, ”education-num” , ”occupation”, ”marital_status”, ”relationship”, ”race”, ”sex”, ”capital-gain”, ”capital-loss”, ”work_hours”, ”native_country”, and ”label”.

In the section, I would like to discuss through each of the feature.

- ✧ Label: There are two kinds of labels here. One is “<=50k”(also “<=50k.”), and we assign that as label “0”. The other is “>50k” (also “>50k.”) and we define it as label ”1”.

- ✧ Age: The feature is a constant distributed from 17 to 90. I tried several ways to deal with the feature. We use the original data to compare with the methods to separate the age into different part (function ”pandas.cut”) and label each part with a value. Then, we standardize the data(function ”scale”) and use Perceptron to see the accuracy in each approach. The results are shown in the following table.

data	label	Accuracy when using Perceptron
Original numeric data		83.11%
[10,20,30,40,50,60,70,80,90]	[1,2,3,4,5,6,7,8]	80.82%
[0,20,40, 60, 80,100]	[1,2,3,4,5]	79.07%
[15,25, 40,65,100]	[1,2,3,4]	80.04%

- ✧ Workclass: There are “Private”, ”Self-emp-not-inc”, ”Self-emp-inc”, ”Federal-gov”, ”Local-gov”, ”State-gov”, ”Without-pay”, ” Never-worked”. Here, we will try some ways to get the different accuracy.

  1. Use original data

Private	7582

Self-emp-not-inc	883
Self-emp-inc	340
Federal-gov	320
Local-gov	675
State-gov	422
Without-pay	6
Never-worked	3
?	622

2. Try to reduce the number of categories: First, we combine “without pay” and “Never-worked” as “No salary”. Second, we put ”Local-gov” and ”State-gov” into “other-gov”. Because the salary to work in ”Federal-gov” might be higher than these two, we keep it as a category by its own and compare with “other-gov”. In the end, we combine “Self-emp-not-inc” and “Self-emp-inc” to “Self-emp”.

Private	7582
Self-emp	1223
Federal-gov	320
other-gov	1097
No-salary	9
?	622

3. After reassigning categories in method 2, replacing “?” with the mode category “Private”.

Private	8204
Self-emp	1223
Federal-gov	320
other-gov	1097
No-salary	9

4. As we can see in method 2, “No salary ” only plays few proportion in the dataset, therefore, we try to combine it with “?” after doing method 2.

Private	7582
Self-emp	1223
Federal-gov	320
other-gov	1097
?	631

After setting each of the method, we use One-Hot encoding (function "pandas.get_dummies") to separate it into different feature. Because it is a string category and has nothing to do with size or value, we won't use integer encoding. In addition, we will not do the standardization after doing One-Hot encoding because one-hot encoding is designed to make the distance in feature space between any two different categorical values, be the same. If we standardize, the relation will no longer hold. The results are present in the following table.

Method	Accuracy when using Perceptron
1.Original data	79.53%
2.Combination stated above (keep "?")	83.11%
3.Replace "?" with the mode after doing method 2	81.61%
4.Combine "No salary " with "?"	79.21%

- Fnlwgt: We only know that the feature is continuous.
- Education: There are "Bachelors", "Some-college", "11th", " HS-grad", " Prof-school", "Assoc-acdm", " Assoc-voc", " 9th", " 7th-8th", "12th", " Masters", "1st-4th", "10th", "Doctorate", " 5th-6th", "Preschool".

  I find that the feature can perfectly correspond to the feature "education-num", therefore, it is a redundant feature and I decide to drop it. If we keep the feature, it will increase the dimensionality of feature space that leads to the worse performance in accuracy as we can see in the following table.

Method	Accuracy when using Perceptron
1.keep "education"	82.24%
2.drop "education"	83.11%

- Education-num: We know that the feature is continuous. We would like to try in 2 ways:

1. Use the original data.
2. Separate it into different numeric labels.

After finishing setting these data, we will standardize it. Then, use Perceptron to get the accuracy results which is shown in the following table.

data	label	Accuracy when using Perceptron
Original numeric data		83.11%
[0,8,11,14,17]	[1,2,3,4]	82.50%
[0,10,12,15, 17]	[1,2,3,4]	81.89%
[0,6,12, 17]	[1,2,3]	81.64%

✧ Occupation: There are “Adm-clerical“, “Armed-Forces“ , “Craft-repair “, “Exec-managerial “, “Farming-fishing “, “Handlers-cleaners “, “Machine-op-inspct “, “Other-service “, “Priv-house-serv “, “Prof-specialty “, “Protective-serv “, “Sales “, “Tech-support “, and “Transport-moving “. We try to reassign them to different category as follows:

1. Use original data.

Adm-clerical	1207
Armed-Forces	4
Craft-repair	1397
Exec-managerial	1347
Farming-fishing	351
Handlers-cleaners	456
Machine-op-inspct	679
Other-service	1124
Priv-house-serv	50
Prof-specialty	1345
Protective-serv	218
Sales	1225
Tech-support	299
Transport-moving	526
?	625

2. Combine “Armed-Forces” and “Priv-house-serv” to “?”, because they only stand small proportion in the data.

3. Replace “?” with mode “Craft-repair” after doing 2.
4. Combine some of them which have closer property.

Service and sales	Sales	2617
	Other-service	
	Priv-house-serv	
	Protective-serv	
Professionals	Tech-support	1644
	Prof-specialty	
Managers	Exec-managerial	1347
Machine-op-inspct	Machine-op-inspct	679
Elementary	Farming-fishing	1333
	Handlers-cleaners	
	Transport-moving	
Craft-repair	Craft-repair	1397
Adm-clerical	Adm-clerical	1207
?	?	629

After setting each of the method, we use One-Hot encoding (function “pandas.get_dummies”) to separate it into different feature. The results are shown as follows.

data	Accuracy when using Perceptron
Method 1:Original data	82.51%
Method 2	82.64%
Method 3	76.42%
Method 4	83.11%

✧ Marital_status: There are several categories inside, including “Married-civ-spouse“, “ Divorced“, “Never-married“, “Separated“, “ Widowed“, “ Married-spouse-absent“, “Married-AF-spouse“. We would like to get the accuracy results in some ways:

1. The original data.

Divorced	1517
Married-AF-spouse	10
Married-civ-spouse	4949
Married-spouse-absent	149
Never-married	3565
Separated	332
Widowed	331

2. First, combine “Married-AF-spouse” and “Married-civ-spouse” to “Married”, then, set “Married-spouse-absent” into “separate”.

Divorced	1517
Married	4959
Never-married	3565
Separated	481
Widowed	331

3. Combine “Married-AF-spouse”, “Married-civ-spouse” and “Married-spouse-absent” to “Married”.

Divorced	1517
Married	5108
Never-married	3565
Separated	332

After setting each of the method, we use One-Hot encoding (function “pandas.get_dummies”) to separate it into different feature. As we stated in “Workclass”, it’s not suitable to use integer encoding. The results are shown as follows.

data	Accuracy when using Perceptron
Method 1:Original data	77.74%
Method 2	83.11%
Method 3	80.32%

✧ Relationship: There are several catagories inside, including “Husband”, “Not-in-family”, “other-relative”, “Own-child”, ”Unmarried”, and “Wife”.

We would try to derive the best result in the following ways:

1. Use the original data.

Husband	4359
Not-in-family	2778
Other-relative	325
Own-child	1700
Unmarried	1167
Wife	524

2. Combine “Husband”, ”Wife”and ”Own-child” to “married”.

Married	6583
Not-in-family	2778
Other-relative	325
Unmarried	1167

3. Combine “Husband” and ”Wife” to “married”.

Married	4883
Not-in-family	2778
Other-relative	325
Own-child	1700
Unmarried	1167

4. Besides the setting in 2, we combine “Not in family” to “unmarried”

Married	6583
Other-relative	325
Unmarried	3945

After setting each of the method, we use One-Hot encoding (function “pandas.get_dummies”) to separate it into different feature. As we stated in “Workclass”, it’s not suitable to use integer encoding. The results are present as follows.

data	Accuracy when using Perceptron
Method 1:Original data	83.11%
Method 2	80.27%

✧ Race: There are “Amer-Indian-Eskimo”, “ Asian-Pac-Islander”, “Black”, “Other”, and “White”. We would like to try in some ways to see if we can get a better accuracy in training data.

1. Use original data

Amer-Indian-Eskimo	123
Asian-Pac-Islander	336
Black	1044
Other	90
White	9260

2. From 1, we can see that “Amer-Indian-Eskimo” only plays a small part in the dataset. Therefore, we try to combine it with “other”.

Asian-Pac-Islander	336
Black	1044
Other	213
White	9260

After finishing setting each of the method, we use One-Hot encoding (function “pandas.get_dummies”) to separate them into different feature. As we stated in “Workclass”, it’s not suitable to use integer encoding. The results are present as follows.

data	Accuracy when using Perceptron
Method 1:Original data	83.11%
Method 2	80.73%
Method 3	76.03%
Method 4	81.29%

- Sex: There are male and female in the feature. We would like to use One-Hot encoding(function “pandas.get_dummies”) to address the feature rather than integer encoding.
- Capital-gain: This is a numeric feature. We will standardize (function ”scale” from sklearn.preprocessing) it and then use it to train the classifier.
- Capital-loss: This is a numeric feature. We will standardize (function ”scale” from sklearn.preprocessing) it and then use it to train the classifier.
- Native-Country: There are several countries in the feature. We find that there’s new country show up in the test data when comparing to the training data. When this situation happened , we will assign it to “?”.

  Because there are many countries in the feature, it will increase the dimensionality a lot when we use One-Hot encoding. Therefore, I would like to try some ways to reduce the list and to see if we can get a better result. The methods are listed as follows:

  1. Use the original data
  2. Assign the countries according to their incomes.
  3. Assign the countries according to their regions.
  4. Assign the countries according to their continents.

After finishing setting each of the method, we use One-Hot encoding (function “pandas.get_dummies”) to separate them into different feature. The results are shown as follows.

data	Accuracy when using Perceptron
Method 1:Original data	81.19%
Method 2 (Incomes)	82.15%
Method 3 (Regions)	83.11%
Method 4 (Continents)	80.60%

After we find the best way to deal with every feature, we separate the numeric features and string features. We would like to see whether it is better to address the numeric features by using “standardization” (function ”scale” from sklearn.preprocessing) or “normalization” (function ”preprocessing.MaxMinScaler()” from sklearn.preprocessing).

data	Accuracy when using Perceptron
Normalization	82.24%
Standardization	83.11%

When talking to dealing with the missing data, we can remove the data, fill it with the mode or mean, or just leave it as “?”. In this dataset, we have missing data only on string features, therefore, we can try to fill with the mode. The results are shown as follows:

3.2. Compensation for unbalanced data

We use SMOTE (function”SMOTE” from imblearn.over_sampling) to address the problem for unbalanced data. We generate the minority class to make the number of each class become equal in training data.

	Before	After
Label 0	8240	8240
Label 1	2613	8240

data	Accuracy when using Perceptron
Remove the data	82.85%
Fill with the mode in string features	75.63%
Keep it as “?”	83.12%

In order to know how classifiers perform after increasing the number of data points, we can compare the accuracy results in training data as the following table.

	Perceptron		SVM		KNN	
	Before	after	Before	after	Before	after
Accuracy	83.76%	80.14%	85.98%	84.85%	84.82%	84.10%

From the table above, we find that all of the accuracies decrease after we add more points to the minority class. In order to have deeper understanding toward each classifier, we look into the classification report. We set the minority (>50K) as our positive label here.

	Perceptron		SVM		KNN	
	before	after	before	after	before	after
precision	0.57	0.46	0.74	0.54	0.70	0.52
recall	0.74	0.89	0.59	0.83	0.57	0.83
F1 score	0.64	0.60	0.65	0.66	0.63	0.64

We know that when F1 score can be an important element for us to decide whether the classifier is good or not. When we compare the F1 score, we find that accuracies decrease in these three algorithms after SMOTE. Hence, we decide to keep it as the original data rather than balance the data.

3.3. Feature extraction and dimensionality adjustment

In this part, I would like to use “sequential backward feature selection” to choose which feature to remove. I would like to drop each of the feature separately to compare the accuracy when using Perceptron before dropping with the one after dropping. (We dropped “education” already when preprocessing because it is a redundant feature as we stated before)

We can see the results from the table given below.

	age	workclass	fnlwgt	capital-loss	work_hours	education-num	sex
keep	81.01%	81.01%	81.01%	81.01%	81.01%	81.01%	81.01%
drop	79.36%	78.13%	79.68%	70.97%	80.97%	72.98%	83.12%
	race	Marital status	occupation	Capital-gain	relationship	native_country	
keep	81.01%	81.01%	81.01%	81.01%	81.01%	81.01%	
drop	81.86%	78.74%	80.06%	73.87%	79.88%	79.04%	

We can find that the accuracy improves when we drop “race” or “sex”. Besides, the accuracy increases more when we drop “sex”. Therefore, we decide to drop “sex” and then, use the same method to test the accuracy again. The results are shown in the following table.

	age	workclass	fnlwgt	capital-loss	work_hours	education-num
keep	83.12%	83.12%	83.12%	83.12%	83.12%	83.12%
drop	83.46%	80.00%	82.26%	81.86%	79.12%	80.57%
	race	Marital status	occupation	Capital-gain	relationship	native_country
keep	83.12%	83.12%	83.12%	83.12%	83.12%	83.12%
drop	83.22%	79.39%	82.91%	74.10%	81.82%	81.64%

We find that when we drop ” age”, the accuracy improves the most this time. So, we decide to drop it and use the same method again.

	fnlwgt	workclass	capital-loss	work_hours	education-num	relationship
keep	83.46%	83.46%	83.46%	83.46%	83.46%	83.46%
drop	79.64%	79.31%	81.86%	80.85%	80.57%	75.51%
	race	Martial status	Capital-gain	occupation	native_country	
keep	83.46%	83.46%	83.46%	83.46%	83.46%	
drop	81.23%	74.15%	81.11%	80.59%	82.07%	

From the above table, there’s no increasing accuracy when we try to drop the left features. To summarize, we drop “age”, ”sex”, and ”education” in the dataset.

3.4. DatasetUsage

I use “adult.train_SMALLER” as my training data and “adult_test” as my test data. Because the performance becomes worse after the correction to the imbalance problem, we decide to keep it as original data..

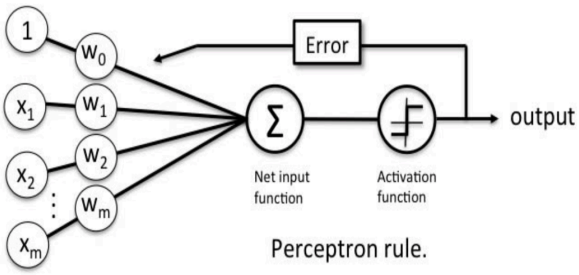
We use all of the data points in “adult.train_SMALLER”, which contains 10,853 points, to train the three classifiers. In order to get the optimal parameters, we use cross validation for two times, one is 5-fold and 1 run in choosing the optimal pair $[\gamma, C]$ in SVM (function ”StratifiedKFold” from sklearn.model_selection), the other is 10 fold and one run in choosing K in KNN(function “cross_val_score” from scikit-learn).

After preprocessing the data, finishing feature selection, and choosing the optimal parameters for each classifier, we use our test data “adult_test”, which includes 16,281 data points, to see how the performances of our classifiers are in the end. Hence, we only use our test set for three times in the whole process.

3.5. Training and Classification

3.5.1 Perceptron

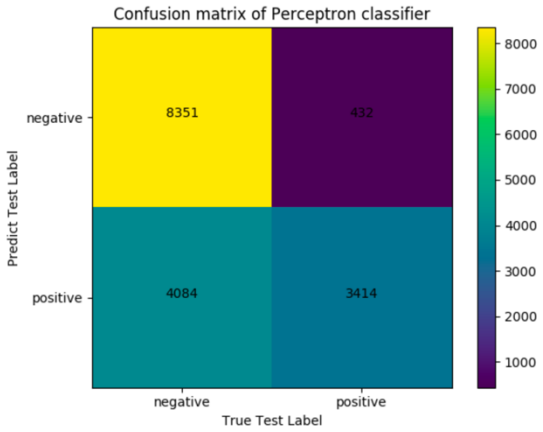
A perceptron is an artificial neuron that can learn and process elements in the training set. The algorithm learns the weight vectors from the inputs data and form the decision boundary for two linearly separable classes. Perceptron receives inputs, adjusts them with certain weight values, and then, when each of the data points in the dataset can be correctly classified or it satisfies our halt conditions, it applies the transformation function to output the final result.[1] The plot showing below can help us understand its operation rule.



Perceptron rule.

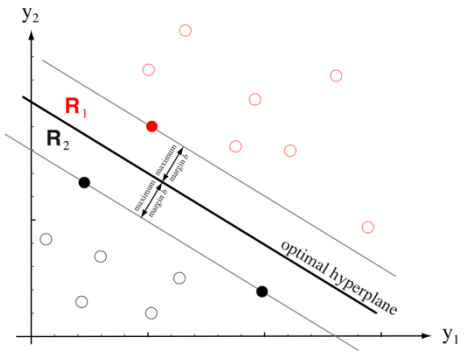
Because the algorithm will correct the weight vectors automatically, what we need to do is to find the initial weight vector. Here, we generate random initial weight vectors (2D array:1 x the number of the features in training data) for 100 times (function “np.random.randn” from numpy). We use these weight vectors to fit with our training data and labels, and then, to find the best accuracy in training data and its corresponding test accuracy. Because the initial weight vector is generated by random and the algorithm shuffle the data each time when we fit the training data, we will get the different best result every time. In my trial this time, the results are shown in the following table.

data	adult.train_SMALLER	adult_test
accuracy	83.76%	83.85%



3.5.2. Support Vector Machine (SVM)

SVM preprocesses the data to represent patterns in a higher dimension. The goal in training a SVM is to find the separating hyperplane with the largest margin. [2] The plot shown below can help us to illustrate the ideal.



In order to derive the best result, we need to tune the parameters in the algorithm. C and Gamma are the parameters for a nonlinear support vector machine with a Gaussian radial basis function kernel. Let's look at C first.

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\sum_i \xi_i$$
$$\text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \forall i$$
$$\xi_i \geq 0, \forall i$$

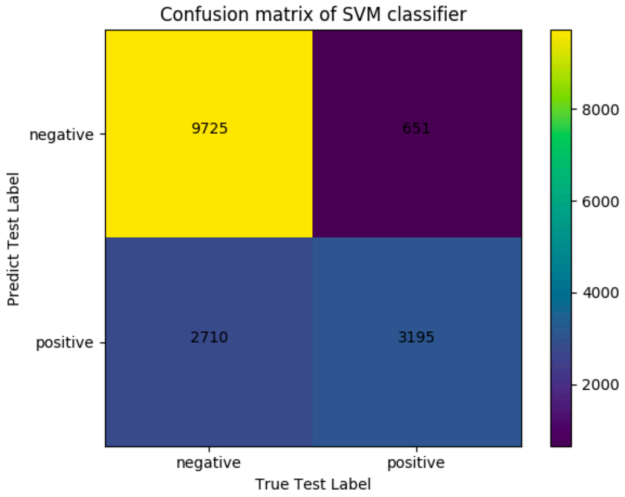
C is a penalty for misclassifying a data point. When C is small, the misclassified data points will not make huge influences to the classifier, so it can accommodate more misclassified data points. When C is large, the classifier is heavily penalized for misclassified data and therefore curves to avoid misclassified data points.

$$K(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2), \gamma > 0$$

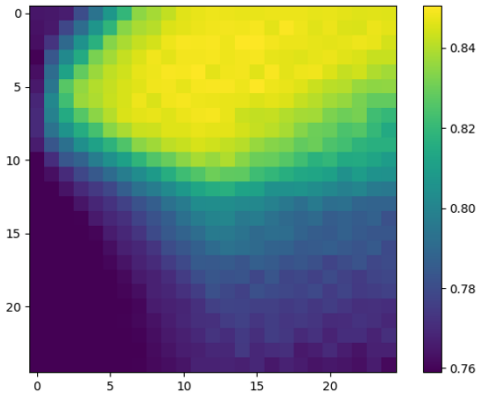
γ is a parameter of the RBF kernel and can be thought as the "spread" of the kernel and therefore the decision region. When γ is low, the decision

region is wider. When γ gets larger, the decision boundary becomes narrower which fits the data points better. However, when γ is too larger as we can see when γ = 500, the decision region looks like circling each of the training data points and causes the overfitting. Therefore, We can know that when we use γ which is too large to test the test dataset, we will get poor accuracy. Here, we use SVM (function "SVC" from sklearn.svm) in Gaussian (RBF) kernel. We pick 25 points for each parameter in the range from $\gamma \in [10^{-2}, 10^{2}]$ and $C \in [10^{-2}, 10^{2}]$ (function "logspace()"). We initialize a 2D array of size $(N_r, N_c) = (25,25)$to store the mean accuracies on the validation set. Also, set up another 2D array to store the estimated standard deviation of accuracies on the validation set. Then, for each pair of [γ, C] perform a 5-fold cross validation (function "StratifiedKFold" from sklearn.model_selection) and store the average accuracy to the corresponding position, and similarly the estimated standard deviation of accuracy. Perform the partitioning randomly but stratified. The best accuracy we get and the corresponding [γ, C] are shown as follows.

[γ, C]=[0.068,3.162]	adult.train_SMALLER	adult_test
accuracy	85.98%	85.32%

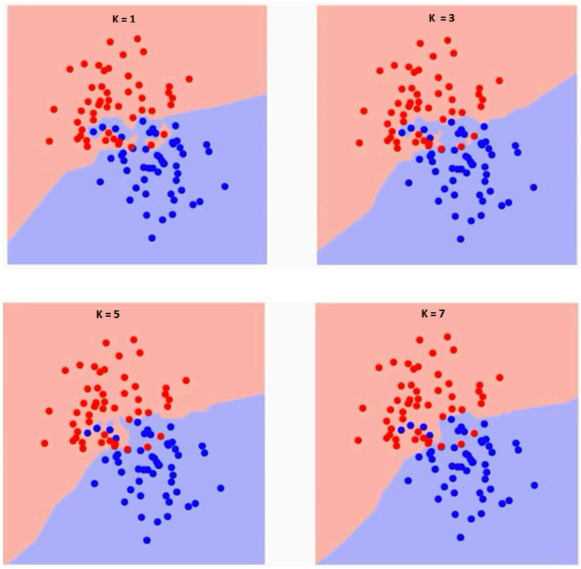


By using different pair of [γ, C] within $\gamma \in [0.01, 100]$ and $C \in [0.01, 100]$, we can get different value of accuracy. The following plot showing the distribution of accuracy when using different pair of [γ, C]. We pick the 25 points within the range we stated before for each parameter. The X-axis follows the order of picking sequence in γ, and the Y-axis follows the order of picking sequence in C.



3.5.3 K-Nearest Neighbors (KNN)

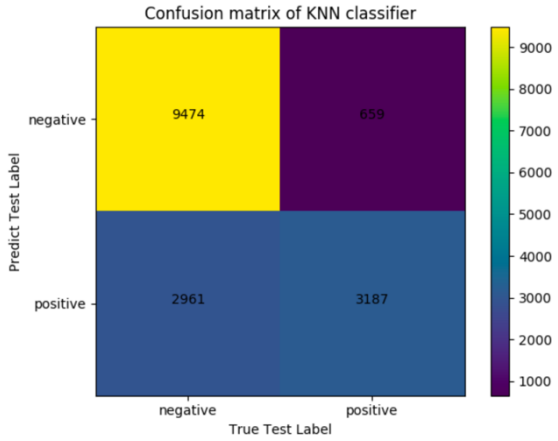
In the algorithm, the output labels depend on the majority labels of K nearest neighbors. We can see an example on how to decide the parameter on Analytics Vidhya[3]. In the plot shown below, we can find that when K is too small, it may lead to overfitting.



When K = 1, the accuracy of training data is almost 100%. This is because the closest point to any training data point is itself, therefore, the prediction is always accurate with K=1.
Because that we can only touch our test data at the end of the process, I would like to use k-fold cross validation[4] to get the accuracy score (function “cross_val_score” from scikit-learn) which involves randomly dividing the training set into k folds, of approximately equal size. The first fold is treated as a validation set, and the method is fit on the remaining (k-1) folds. The misclassification rate is then computed on the observations in the held-out fold. This procedure is repeated for k times; each time, a different group of observations is treated as a validation set. This process results in k estimates of the test error which are then averaged out. Here, I will perform 10 folds and try to find the optimal K in range 1 to 30. We find

that the optimal K=23 with cross validation score=83.47%. The results for "adult.train_SMALLER" and "adult_test" are shown as following table.

data	adult.train_SMALLER	adult_test
accuracy	84.82%	84.12%



4. Comparison, Results, and Interpretation

	Perceptron		SVM		KNN	
	train	test	train	test	train	test
Accuracy	83.76%	83.85%	85.98%	85.32%	84.82%	84.12%

As we can see from the above table, we get our best results in SVM, which are 85.98% in training data and 85.32% in testing data. And all of the accuracy results are larger than the baseline.

In the data set, the baseline $= \frac{8240}{8240+2613} x100\% = 75.92\%$

	Perceptron			**SVM [γ, C]=[0.068,3.162]**			**KNN K=23**		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Label 0 **(≤50K)**	0.88	0.87	0.88	0.88	0.93	0.91	0.87	0.93	0.90
Label 1 **(>50K)**	0.6	0.61	0.6	0.74	0.59	0.65	0.7	0.57	0.63

From the table above, we can see the performance of our classifiers from F1 score in test data that SVM > KNN > Perceptron. (We view the minority ">50K" as our positive label which is label 1 here.) Besides, we can see that the big difference between label 0 and label 1 in each of the value. The reason it happens is that the imbalance number of the two classes. This phenomenon can be improved when we

balance the number of the two classes. However, it will reduce the accuracy of the classifier.

5. Summary and conclusions

In the project, we try to deal with the raw data, which might suffer from missing data, imbalance data, too many dimensionalities in feature space, etc. We can use trials and errors to categorize our items in one feature to see if we can get a better result, apply feature selection to decide which feature to drop, use cross-validation to choose the optimal parameters for the classifier, …and so on. There are many ways to improve the accuracy. Sometimes, when we find a problem trying to fix it and thought that the accuracy can be improved after the correction. However, the situation is not like what we had thought. “Dealing with imbalance data” is one of the examples to illustrate the ideal. When we see the ratio of the number of classes which is approximately 4:1, we might think that the data suffers from imbalance problem. We oversample the minority class to keep the number balance. However, the accuracy drops after the correction. To make a conclusion, we can only improve the accuracy after continuous trials and errors because it’s a trade-off problem sometimes. In the project, we find that all of the features play important roles in a person’s income per year except “age”,” education”, and “sex”.

References

[1] ” Perceptron Tutorial” [online]

Available: https://www.simplilearn.com/what-is-perceptron-tutorial

[2] Richard O. Duda, Peter E. Hart, and David G. Stock, ”Support Vector Machine,” in ”Pattern Classification ”: Second Edition.

[3] TAVISH SRIVASTAVA, “Introduction to k-Nearest Neighbors: Simplified (with implementation in Python),” 27th May 2018 [online]

Available: https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/

[4] Kevin, “A Complete Guide to K-Nearest-Neighbors with Applications in Python and R,” 13 Jul 2016 [online]

Available: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/