Chalmers Tekniska Högskola

CHALMERS
UNIVERSITY OF TECHNOLOGY

DESIGN OF AI SYSTEMS (DAT410)

Module 3: AI tools

Group 38

Himanshu Sahni
9812075498
sahni@chalmers.se
MPCAS

Nina Uljanić
9609134920
uljanic@chalmers.se
MPCAS

We hereby declare that we have both actively participated in solving every exercise. All solutions are entirely our own work, without having taken part in other solutions.

Hours:
Himanshu Sahni: 22h
Nina Uljanić: 22h

1. Reading and reflection

Himanshu

The paper "Hidden Technical Debt in Machine learning Systems" introduces technical debt as a major concern in Machine learning. It can be defined as the result achieved when an easy solution is deployed instead of an appropriate solution. It is also referred to as design debt or code debt. The machine learning systems have a lot of low-level design and thus they are difficult to maintain. Therefore, it becomes crucial to develop some policies with which we can ensemble future improvements. The technical debt in the ML systems is difficult to detect as it is located at the system level rather than the code level.
Some of the common causes of eroding boundaries are explained here: The process of mixing signals together in an ML system is known as entanglement. It can be explained as every bit of data is related, therefore a small change can affect the entire system. This is usually related to the CACE principle (Changing Anything Changes Everything). Some of the strategies to deal with entanglements are to isolate models and serve ensembles and focus on detecting changes in prediction behavior. Correction cascades refer to the linkage between the models and the technical debt generated due to the link. The method to mitigate correction cascades is to learn the corrections directly within the model and create separate models. Ml systems generate data that can be used for different purposes but if the use case is not defined properly and linked then it becomes an undeclared consumer. One of the ways to handle this is to restrict access or use service-level agreements.
In ML systems, data dependencies carry a similar capacity for building debt but may be more difficult to detect whereas the code dependencies can be identified via static analysis by compilers and linkers. Unstable data dependencies are known as direct consumption of a data source and can change due to different factors and these changes can affect the output as well. A common strategy to mitigate or avoid this effect is to create a versioned copy. Underutilized data dependencies are defined as the input signals that add little incremental modeling benefit and can affect the model presented in the following ways:

- 1. Legacy features - Early features that affect future data.
- 2. Bundled Features - This occurs due to the inclusion of unnecessary features.
- 3. E-features - overfitting due to excess improvements.
- 4. Correlated features - This happens due to incorrect measurements.

ML systems are related to a form of analysis of debt and influence their own behavior. The two known types are the direct feedback loops and the hidden feedback loops. The direct feedback loops directly influence the selection of its own future training data. It is possible to mitigate these effects by using some amount of randomization or by isolating certain parts of data from being influenced. A hidden feedback loop is one in which two systems influence each other indirectly throughout the world.

Configuration debt implies that there is an accumulation of incorrect or insufficient configurations. Some of the principles of a good configuration are:

- 1. It should be easy to specify a configuration as a small change compared to the previous configuration.
- 2. It should be hard to make manual errors, omissions, or oversights.
- 3. It should be easy to see visually the difference between the two configurations.

- 4. It should be easy to automatically assess and verify the basic facts about the configuration.
- 5. Configurations should undergo a full code review and be checked into a repository.

As the external world is unstable it creates maintenance costs. Fixed THresholds in a dynamic environment - it is common to manually set a particular action within a model and as the input data changes, it may be required to change the actions required for the model. Monitoring and testing - It is required to have global and comprehensive testing for ML systems. It is important to know the variables or parameters that are to be monitored. For example, in prediction we observe the distributed labels and compare them with the distribution of observed labels as this might be helpful in detecting the changes in world behavior. Action tools can be used as a powerful but simple tools related to notifications.
Some of the other areas of ML-related debt are:

- 1. Data testing debt
- 2. Reproducibility Debt
- 3. Process management debt referred to multi-tasking and the existence of several processes interacting in different interfaces.
- 4. Cultural Debt refers to creating team cultures that reward deletion of features, reduction of complexity, improvements in reproducibility, stability, and monitoring to the same degree that improvements in accuracy are valued

We can conclude that engineers and researchers must be mindful of the problem of technical debt. In practical terms, having the awareness of technical debt in any of the possible terms is extremely important and can turn out to be an asset. The research solutions that deliver a minor boost in accuracy at the expense of large system complexity are rarely wise, even the inclusion of a couple of apparently harmless data dependencies might block development.

Nina

The metaphor of *technical debt* used in software engineering refers to the trade-offs made by software developers when developing a minimum viable product (MVP) which would be fixed and/or improved at a later time; the quality of the product is sacrificed (for the time being) in order to develop a product that satisfies a minimum number of requirements. This sacrifice may include planning, design, and documentation. Perfecting the product comes later; this work that is left for later is what the metaphor *technical debt* refers to.

The same metaphor can be applied to the development of ML systems. ML systems are relatively cheap to develop and deploy, however maintaining them over time is difficult and expensive. In a ML system, only a tiny fraction of the code is used for learning and for prediction; the rest is made of "traditional" code. That is, ML systems have the same problems as traditional software, as well as ML-specific issues. Therefore, it can be said that ML systems are especially prone to incurring technical debt. Ideally, one would want to minimize the costs of maintenance and also the complexity of the system - and this can be done by following a set of specific practices that simplify the design of the system, consequently lowering the *technical debt*.

Technical debt is not necessarily a bad thing, so long it is managed carefully. However, all debt needs to be resolved at some point. This should be done rather sooner than later because the *technical debt* accumulates over time. Hidden debt compounds silently, i.e. it might be noticed very late, with grave consequences.

The paper identifies a set of ML-specific risk factors. They are divided into seven categories, however naming all of the risk factors here would make this summary much longer than (about) half a page - two pages, in fact. As a software engineer, I have found it interesting to see how the same issues that plague software engineers also trouble ML engineers.

2. Implementation

Using Pandas we imported the .csv files as Pandas data frames, which are simple to work with. We created five different variables - one for each of the five cities. We then created a scatter matrix of the Beijing and Shenyang data sets to visualize the relation between the various parameters, with regard to the *PM_HIGH* component. This was not mandatory, but we thought it would be interesting to see a visual representation of the data we are working with.

Next, we created a class called Classifier which is based on the KMeans algorithm. It is an unsupervised learning algorithm used to perform clustering on the input data and then it maps them into the cluster labels. We have used two libraries in building this function i.e., Counter and KMeans. We have defined a class called Classifier which creates an instance of the K-means clustering algorithm and we can specify the number of clusters in it.

The class contains four functions: fit, predict, score, and get cluster labels.

- First is the *fit* function, which takes the training and validation data and sets them as its (the model's) knowledge base. It fits the clustering algorithm to the input data X and maps the resulting clusters to class labels using the get cluster labels method. Here, X is the input data which is a matrix of the various features.

- Next comes the *predict* function, which takes the input X, and predicts cluster labels for the feature vectors using the clustering algorithm. Basically, it maps the vectors to the closest clusters and returns the class labels.

- The *score* method takes a matrix of feature vectors X and their corresponding true class labels y, and returns the mean accuracy of the classifier on this input data. The accuracy is calculated by comparing the predicted class labels to the true class labels.

- The *get cluster labels* method maps the clusters generated by the clustering algorithm to class labels. It does this by first counting the number of feature vectors assigned to each cluster, and then calculating the sum of the true class labels for each feature vector assigned to each cluster. Finally, it calculates the average class label for each cluster and rounds it to the nearest integer value. The resulting mapping of clusters to class labels is returned as a dictionary.

After implementing the methods, we divided the data set into training and validation sets, and then we merged the training data for Beijing and Shenyang into one data frame, as well as validation data for the two cities. We preprocessed the data contained in the two new merged data frames. After preprocessing, we ran 10 iterations of evaluations of the training and validation data in order to find the accuracy of our model, as well as the accuracy of scikit-learn implementation on the two datasets. This was done for comparison. The accuracy of our model is at 72% for training data and 73% for validation data, compared to scikit-learn's 51% and 50%, respectively.

Next, we extracted and preprocessed the data for Guangzhou and Shanghai and evaluated the model in the same manner as described above. We found that the validation accuracy of our model is at 93% for Guangzhou and 90% for Shanghai, while scikit-learn's accuracy was at 61% for Guangzhou and 63% for Shanghai.

We can see that our model performed better when evaluated on the training dataset than when evaluated on the validation dataset. However, there was not much of a difference between our implementation and the sci-kit-learn implementation. This leads us to believe that our implementation is working well. However, it is still difficult for us to interpret the reason for such low accuracy when we use the sci-kit learn library.

3. Discussion

In most cases, the accuracy of a system when deployed is lower than the accuracy measured during development (training). Was this the case for your system? What are the possible reasons for this? Discuss which assumptions are made during development that may not hold, and what could be done to improve the system. Do these assumptions hold for the problem above?

Our implementation achieved the following accuracy: 72% for training and 73% for validation with regards to Beijing and Shenyang, 93% for Guangzhou, and 90% for Shanghai. Indeed, the accuracy of our system was higher when evaluated on the training data compared to validation data (93% vs 90%).

There can be several reasons why an ML system may experience a performance dip when it is evaluated in deployment compared to during development.
We have identified some of the most likely causes and have described them below:

- Overfitting occurs when the model has learned to fit the training data "too well." This means that it has captured both the signal - the underlying patterns or relationships in the data, and the noise - irrelevant patterns or random fluctuation in the data. In other words, overfitting is equal to memorizing the training data instead of learning the underlying patterns that characterize the dataset. This leads to poor performance of the model when applied to new, unseen data.

- Shift in the distribution of the data refers to the difference in data used for development and deployment. This shift in distribution can occur due to changes in the data generating process, data collection method, or due to changes in the data itself, among others. The issue is that the model may not be equipped to handle the new distribution of data, resulting in decreased performance. To mitigate this issue, the model can be allowed to learn using online methods, however, this can be dangerous and should be applied with care.

- Incorrectly choosing an ML approach to address the issue can have a significant impact on accuracy. Therefore, it is important to try out different approaches and then choose a set of techniques that are suited to address the given challenge in terms of efficiency and efficacy. Also, it is crucial to tune the parameters properly even if the technique chosen is correct.

Discuss which assumptions are made during development that may not hold
It is likely that our model suffers from the latter. China is a vast country, with a large land area, and the four cities above are spread throughout the country. While Beijing and Shenyang, the two cities whose data was used for training, are located in the north of China and are fairly near in terms of geographical distance, Guanzhou is located in the southern region of China, and Shanghai is located halfway between these two regions. These cities are all located in different climate regions, and the model has been trained on data representing only one of the regions. It is understandable that the model would not be as efficient in predicting the results for cities located in other climate zones. The model may need to be retrained or adapted to the new data if it is to perform well.

Individual Summary and Reflection

Lecture 3: AI Tools (2023-01-31)

Lecture summary by Himanshu Sahni

In the previous lecture, we learned about the various AI tools that can be used to develop AI systems. They can be selected based on the nature of the system, however, they represent part of the ML lifecycle: 1. Problem definition and data collection 2. Data representation processing 3. Modeling 4. Learning/optimization 5. Evaluation. Usually, the data is stored in terms of matrices and vectors because it is easy to apply ML techniques to them. These structures of data in terms of matrices are called data frames and they can be used to visualize the given data in various representations. It is usually preferred to have ordered data as it reduces the computational cost. However, in real life, we face the problem of missing values or inconsistencies. There are techniques to handle these missing values by finding correlations within the data, for example, imputation (reconstructing data with known values) and informative missingness (reconstructing the data using the known positions of unknown values and available data). The structure of the systems consists of three main functions i.e., fit, predict, and score (evaluate). The fit function trains and stores the model parameters that maximize the objective. The predict function is used to predict the outcome and the score function is used to check the predictions and compare them with the actual value and finally tells the score (accuracy) of the system. It is always preferred to do some preprocessing i.e., standardizing features (scaling) on data before training, these help in bringing uniformity in the data and reduce bugs. We also learned about differential systems, these are the systems where the objective function is differential in terms of parameters θ. Also, we looked into some optimization tools for constrained and non-differential problems like CVX and Gurobi. These tools provide syntax for optimization problems and solvers as well. In the end, we looked into different ways of model evaluation like RMSE, k-fold cross-validation, and some tools like tensorboard which help in tracking the experiment's progress.

Module 2 reflection by Himanshu Sahni

In the previous module, I got an in-depth understanding of recommender systems i.e, types of recommender systems, how they work, and the problems that are faced while developing a recommender system. It is extremely important to define the system and its features precisely because this leads to saving a lot of time and resources. In the previous module, we looked at how to handle data, for example, the inconsistencies and missing values in it. Sometimes the recommender system faces the cold start problem initially and this could lead to generalization errors. It can also happen when the required techniques are not applied to the input data because this can affect the training and eventually the feedback is affected. The most important learning for me is that it is important to have a good quantity and quality of data to build a recommender system. Also, having a large amount of data does not always imply that the data is relevant or that it can be processed or converted into meaningful data. The ratio between the relevant data to the total amount of data available in the selected proxies should be balanced to accurately identify certain attributes, which will lead to improving the performance of the recommendation system.

Lecture summary by Nina Uljanić

Contrary to what one might think when one hears "AI tools", AI tools do not fall under the category of AI; instead, they are tools (software applications) that are used to build AI systems. In this lecutre, a number of such tools was mentioned, with a greater emphasis put on tools used in statistical machine learning. The most commonly used programming tools (ML frameworks) were mentioned: scikit-learn, PyTorch, TensorFlow and Keras. These frameworks provide a range of functionality (algorithms) for building and training machine learning models - classification, regression, dimensionality reduction, etc For those who do not intend to implement ML models from scratch, these tools provide a simple way of building ML systems.

The data used in AI systems is usually stored in tables (matrices, arrays) and linear algebra provides mathematical operations for working with this data. Two libraries, Pandas and NumPy are commonly used for data analysis and numerical computing. For example, Pandas stores the tables as dataframes that maintain the labels for rows and columns, making it easier to work with data. The aforementioned AI tools are built on top of such these libraries, providing a convenient and efficient way to perform linear algebra calculations. This is of great use to anyone developing an AI system, as AI/ML algorithms rely heavily on linear algebra.

Module 2 reflection by Nina Uljanić

In module 2 we have implemented a recommendation system for movies. Working on implementing such a system has taught me that careful consideration of the design is required in order to create a well-performing recommendation system - there are many parameters which need to be taken into consideration. I have also learned that there exist two different approaches to building recommendation systems: collaborative and content filtering. The former approach recommends items to a user based on the preferences of similar users, while the latter uses the attributes of items to recommend items that are most similar to the items a user has liked in the past instead of relying on other, similar users. Additionally, I was made aware that incomplete data commonly occurs in such a system, which may be an issue depending on the implementation.

Importing Necessary libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score
```

In [2]:
```
df_beijing=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Beijing_labeled.csv")
df_chengdu=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Chengdu_labeled.csv")
df_guangzhou=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Guangzhou_labeled.csv")
df_shanghai=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Shanghai_labeled.csv")
df_shenyang=pd.read_csv("C:\\Users\\hsahn\\Design of AI systems\\module3\\Shenyang_labeled.csv")
```

In [3]:
```
df_beijing.info() # zero null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2071 entries, 0 to 2070
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   season         2071 non-null   int64  
 1   DEWP           2071 non-null   float64
 2   HUMI           2071 non-null   float64
 3   PRES           2071 non-null   float64
 4   TEMP           2071 non-null   float64
 5   Iws            2071 non-null   float64
 6   precipitation  2071 non-null   float64
 7   cbwd_NE        2071 non-null   int64  
 8   cbwd_NW        2071 non-null   int64  
 9   cbwd_SE        2071 non-null   int64  
 10  PM_HIGH        2071 non-null   float64
dtypes: float64(7), int64(4)
memory usage: 178.1 KB
```

In [4]:
```
df_shenyang.info() # zero null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   season         824 non-null    int64  
 1   DEWP           824 non-null    float64
 2   HUMI           824 non-null    float64
 3   PRES           824 non-null    float64
 4   TEMP           824 non-null    float64
 5   Iws            824 non-null    float64
 6   precipitation  824 non-null    float64
 7   cbwd_NE        824 non-null    int64  
 8   cbwd_NW        824 non-null    int64  
 9   cbwd_SE        824 non-null    int64  
 10  PM_HIGH        824 non-null    float64
dtypes: float64(7), int64(4)
memory usage: 70.9 KB
```

In [5]:
```
df_guangzhou.info() # zero null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1352 entries, 0 to 1351
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   season         1352 non-null   float64
 1   DEWP           1352 non-null   float64
 2   HUMI           1352 non-null   float64
```

```
 3   PRES           1352 non-null   float64
 4   TEMP           1352 non-null   float64
 5   Iws            1352 non-null   float64
 6   precipitation  1352 non-null   float64
 7   cbwd_NE        1352 non-null   int64
 8   cbwd_NW        1352 non-null   int64
 9   cbwd_SE        1352 non-null   int64
 10  PM_HIGH        1352 non-null   float64
dtypes: float64(8), int64(3)
memory usage: 116.3 KB
```

In [6]:
```
df_shanghai.info() # zero null values
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1351 entries, 0 to 1350
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   season         1351 non-null   int64
 1   DEWP           1351 non-null   float64
 2   HUMI           1351 non-null   float64
 3   PRES           1351 non-null   float64
 4   TEMP           1351 non-null   float64
 5   Iws            1351 non-null   float64
 6   precipitation  1351 non-null   float64
 7   cbwd_NE        1351 non-null   int64
 8   cbwd_NW        1351 non-null   int64
 9   cbwd_SE        1351 non-null   int64
 10  PM_HIGH        1351 non-null   float64
dtypes: float64(7), int64(4)
memory usage: 116.2 KB
```

In [7]:
```
df_beijing.head()
```

Out[7]:

	season	DEWP	HUMI	PRES	TEMP	Iws	precipitation	cbwd_NE	cbwd_NW	cbwd_SE	PM_HIGH
0	4	-8.0	79.0	1026.0	-5.0	23.69	0.0	0	0	1	1.0
1	4	-11.0	85.0	1021.0	-9.0	105.93	1.1	0	0	1	0.0
2	4	-21.0	43.0	1030.0	-11.0	117.55	0.0	0	1	0	0.0
3	4	-25.0	33.0	1034.0	-12.0	39.35	0.0	1	0	0	0.0
4	4	-24.0	30.0	1034.0	-10.0	59.00	0.0	1	0	0	0.0

In [8]:
```
df_beijing.describe()
```

Out[8]:

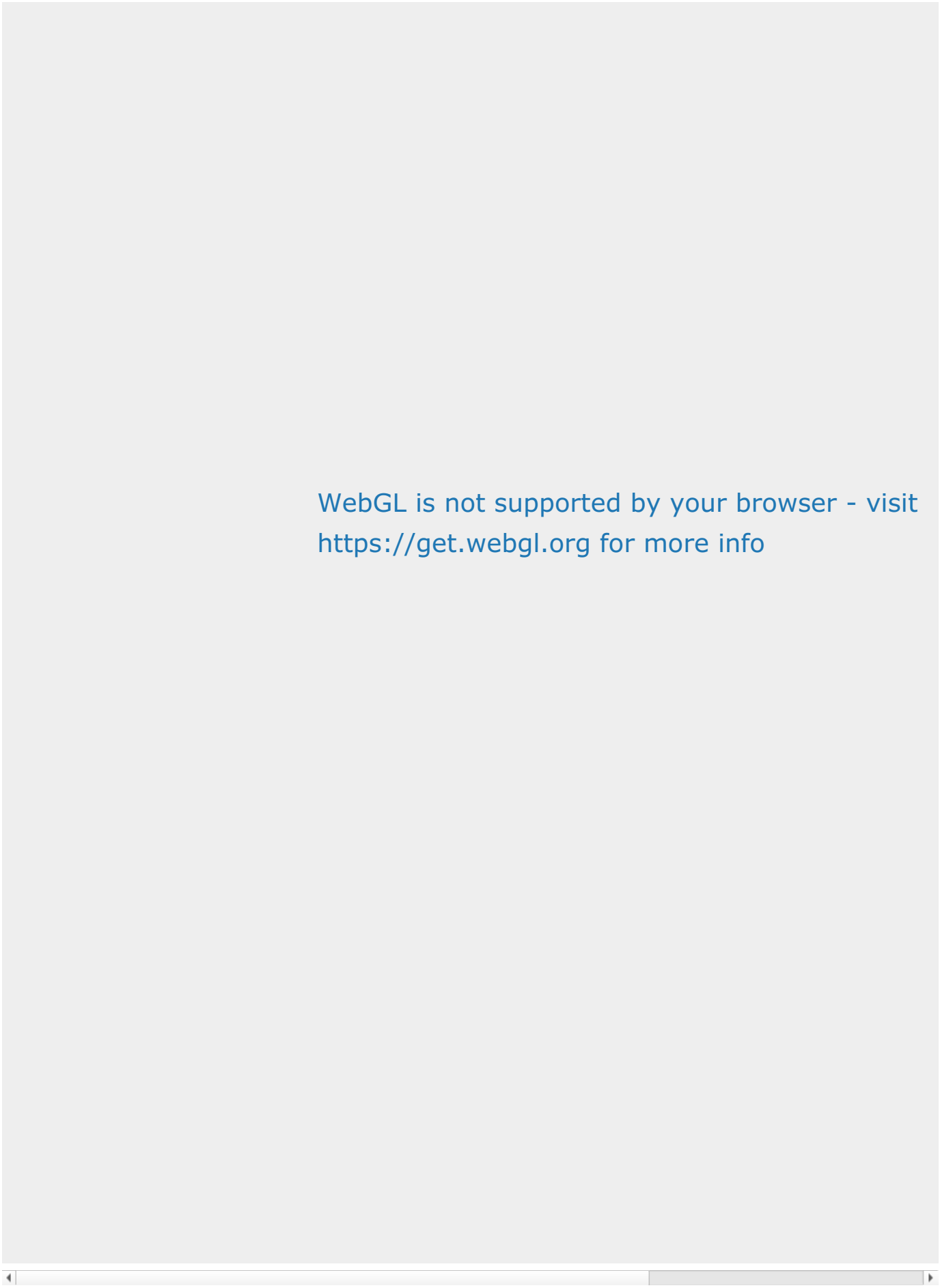
	season	DEWP	HUMI	PRES	TEMP	Iws	precipitation	cbwd_NE	cbwd_NW	cbwd_SE	P
count	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	2071.000000	207
mean	2.485756	1.201835	41.121197	1016.711733	16.220666	21.499034	0.046548	0.113955	0.306132	0.315789	
std	1.123601	14.617897	22.833175	10.255643	11.688141	47.630452	0.495941	0.317833	0.460997	0.464942	
min	1.000000	-36.000000	3.000000	994.000000	-12.000000	0.450000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	-11.000000	22.000000	1008.000000	5.000000	1.790000	0.000000	0.000000	0.000000	0.000000	
50%	2.000000	1.000000	37.000000	1016.000000	18.000000	4.470000	0.000000	0.000000	0.000000	0.000000	
75%	3.500000	15.000000	57.000000	1025.000000	27.000000	16.990000	0.000000	0.000000	1.000000	1.000000	
max	4.000000	27.000000	100.000000	1044.000000	39.000000	468.050000	14.700000	1.000000	1.000000	1.000000	

In [9]:
```
df_beijing.shape
```

Out[9]: (2071, 11)

scatter matrix (showing relation between each variable)

In [10]:
```
import plotly.express as px
fig = px.scatter_matrix(df_beijing,color = 'PM_HIGH',  title="Beijing")
fig.update_layout(
    width=1200,
    height=1200
)
fig.show()
```



In [11]:
```
import plotly.express as px
```

```
fig = px.scatter_matrix(df_shenyang,color = 'PM_HIGH',  title="Shenyang")
fig.update_layout(
    width=1200,
    height=1200
)
fig.show()
```

Creating datframes to input in our model

In [12]:
```
X_beijing = df_beijing.iloc[:, :10]
y_beijing = df_beijing.iloc[:,-1]
X_shenyang = df_shenyang.iloc[:, :10]
y_shenyang = df_shenyang.iloc[:,-1]
```

In [13]:
```
# X_beijing.head()
# y_beijing.head()
# X_shenyang.head()
# y_shenyang.head()
# X_beijing.shape
```

Splitting the datasets into train and validation sets

In [14]:
```
X_beijing_train, X_beijing_validation, y_beijing_train, y_beijing_validation = train_test_split( X_beijing, y_bei
X_shenyang_train, X_shenyang_validation, y_shenyang_train, y_shenyang_validation = train_test_split( X_shenyang,
```

In [15]:
```
# X_beijing_train.shape
# X_beijing_validation.shape
# X_shenyang_train.shape
# X_shenyang_validation.shape
```

Merging the train and test sets for the two cities

In [16]:
```
# Training
X_beijing_shenyang_train = pd.concat([X_beijing_train, X_shenyang_train],ignore_index = True, sort = False)
y_beijing_shenyang_train = pd.concat([y_beijing_train, y_shenyang_train],ignore_index = True, sort = False)
```

In [17]:
```
# X_beijing_shenyang_train.shape
# y_beijing_shenyang_train.shape
```

In [18]:
```
# Test
X_beijing_shenyang_validation = pd.concat([X_beijing_validation, X_shenyang_validation],ignore_index = True, sort
y_beijing_shenyang_validation = pd.concat([y_beijing_validation, y_shenyang_validation],ignore_index = True, sort
```

In [19]:
```
# X_beijing_shenyang_validation.shape
# y_beijing_shenyang_validation.shape
```

preprocessing (scaling the data)

In [20]:
```
X_beijing_shenyang_train = preprocessing.StandardScaler().fit_transform(X_beijing_shenyang_train)
X_beijing_shenyang_validation = preprocessing.StandardScaler().fit_transform(X_beijing_shenyang_validation)
```

In [21]:
```
# X_beijing_shenyang_train.shape
# y_beijing_shenyang_train.shape
# X_beijing_shenyang_validation.shape
# y_beijing_shenyang_validation.shape
```

Using the Elbow method to find the optimal number of clusters

In [22]:
```
center = []
for k in range(1,11):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(X_beijing_shenyang_train)
    center.append([k,kmeans.inertia_])
center = pd.DataFrame(center, columns=["number_of_clusters","distance"])
```

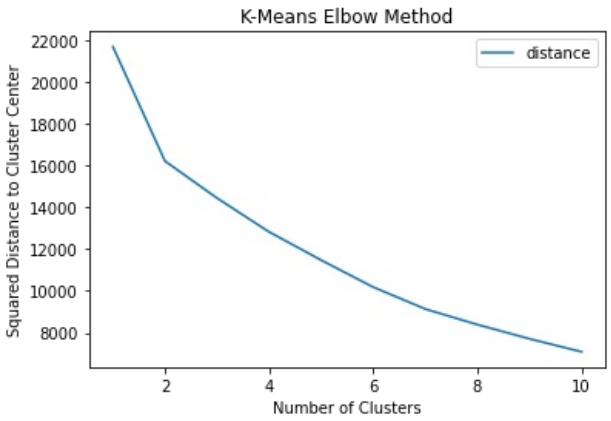
```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:
```

```
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

In [23]:
```
center.set_index("number_of_clusters").plot()
plt.xlabel("Number of Clusters")
plt.ylabel("Squared Distance to Cluster Center")
plt.title("K-Means Elbow Method")
```

Out[23]:
```
Text(0.5, 1.0, 'K-Means Elbow Method')
```



We can see that the squared distance is small for more number of clusters. However in our case we have only predictions as 0 and 1 so we will use number of clusters = 2 for our implmentation.

Using Scikit learn library

In [24]:
```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X_beijing_shenyang_train)
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

Out[24]:
```
KMeans(n_clusters=2)
```

In [25]:
```
y_pred1 = kmeans.predict(X_beijing_shenyang_train)
score = accuracy_score(y_beijing_shenyang_train, y_pred1)
print(score)
```

```
0.5080608014739751
```

In [26]:
```
y_pred2 = kmeans.predict(X_beijing_shenyang_validation)
score = accuracy_score(y_beijing_shenyang_validation, y_pred2)
print(score)
```

```
0.5013812154696132
```

In [27]:
```
# Guanzhou
X_guanzhou = df_guangzhou.iloc[:, :10]
y_guanzhou = df_guangzhou.iloc[:,-1]

# Shanghai
X_shanghai = df_shanghai.iloc[:, :10]
y_shanghai = df_shanghai.iloc[:,-1]

# X_guanzhou.head()
```

In [28]:
```
X_guanzhou = preprocessing.StandardScaler().fit_transform(X_guanzhou)
X_shanghai = preprocessing.StandardScaler().fit_transform(X_shanghai)
```

In [29]:
```
y_pred3 = kmeans.predict(X_guanzhou)
score = accuracy_score(y_guanzhou, y_pred3)
print(score)
```

```
0.617603550295858
```

In [30]:
```
y_pred4 = kmeans.predict(X_shanghai)
score = accuracy_score(y_shanghai, y_pred4)
print(score)
```

```
0.6365655070318282
```

Using our Implementation - Creating Class and required functions for KNN model

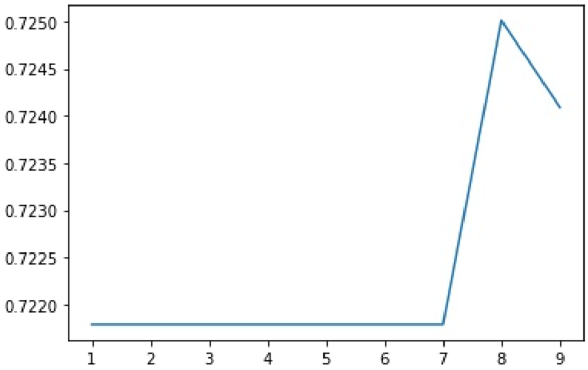
In [31]:
```
class Classifier:
    def __init__(self, n_clusters):
        self.n_clusters = n_clusters
        self.model = KMeans(n_clusters=n_clusters)

    def fit(self, X, y):
        self.model.fit(X)
        self.cluster_labels = self._get_cluster_labels(X, y)

    def predict(self, X):
        clusters = self.model.predict(X)
        return np.array([self.cluster_labels[cluster] for cluster in clusters])

    def score(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred == y)

    def _get_cluster_labels(self, X, y):
        clusters = self.model.predict(X)
        cluster_counts = Counter(clusters)
        cluster_sums = Counter()
        for i, cluster in enumerate(clusters):
            cluster_sums[cluster] += y[i]
        cluster_labels = {}
        for cluster in range(self.n_clusters):
            cluster_labels[cluster] = round(cluster_sums[cluster] / cluster_counts[cluster])
        return cluster_labels
```

In [32]:
```
scores_training = []
for k in range(1, 10):
    classifier = Classifier(k)
    classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scores_training.append(classifier.score(X_beijing_shenyang_train, y_beijing_shenyang_train))
# print(scores)
plt.plot(range(1, 10), scores_training)
plt.show()
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
```

```
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:
```

```
KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```



In [33]:

```python
classifier = Classifier(2)
classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
print(classifier.score(X_beijing_shenyang_train, y_beijing_shenyang_train)) # Training accuracy
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

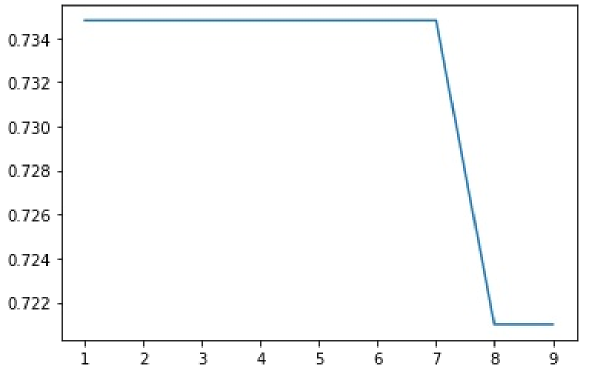
```
0.7217871948410871
```

In [34]:

```python
scores_validation = []
for k in range(1, 10):
    classifier = Classifier(k)
    classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scores_validation.append(classifier.score(X_beijing_shenyang_validation, y_beijing_shenyang_validation))
plt.plot(range(1, 10), scores_validation)
plt.show()
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:
```

```
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```



In [35]:

```
classifier = Classifier(2)
classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
print(classifier.score(X_beijing_shenyang_validation, y_beijing_shenyang_validation)) # Validation accuracy
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:
```

```
The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

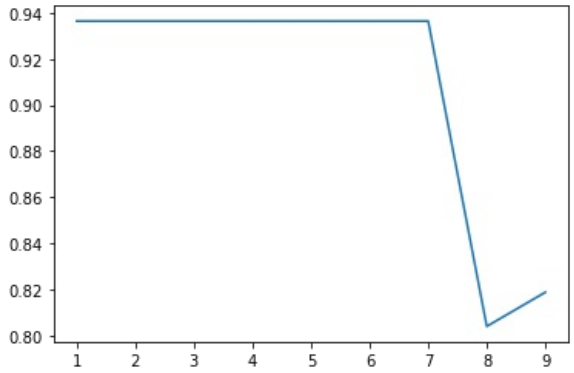
```
0.7348066298342542
```

In [36]:

```
scores_guanzhou = []
for k in range(1, 10):
    classifier = Classifier(k)
    classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scores_guanzhou.append(classifier.score(X_guanzhou, y_guanzhou))
plt.plot(range(1, 10), scores_guanzhou)
plt.show()
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```



In [37]:

```
classifier = Classifier(2)
classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
print(classifier.score(X_guanzhou, y_guanzhou)) # Test accuracy for Guanzhou
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

```
0.9363905325443787
```

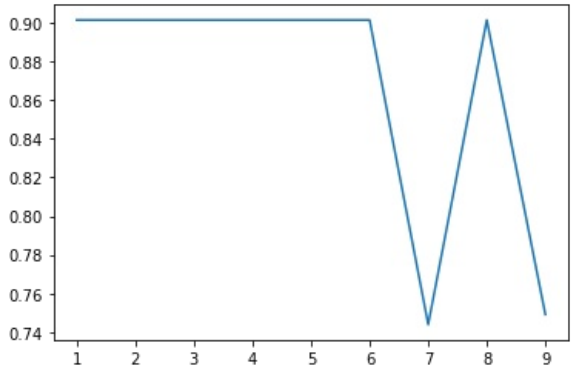
In [38]:

```
scores_shanghai = []
for k in range(1, 10):
    classifier = Classifier(k)
    classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
    scores_shanghai.append(classifier.score(X_shanghai, y_shanghai))
plt.plot(range(1, 10), scores_shanghai)
plt.show()
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
```

```
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```



In [39]:

```
classifier = Classifier(2)
classifier.fit(X_beijing_shenyang_train, y_beijing_shenyang_train)
print(classifier.score(X_shanghai, y_shanghai)) # Test accuracy for Guanzhou
```

```
C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: FutureWarning:

The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppr
ess the warning

C:\Users\hsahn\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You
can avoid it by setting the environment variable OMP_NUM_THREADS=9.
```

```
0.9015544041450777
```

In [ ]: