City Prediction from Weather Data

July 9, 2019

1 Introduction

We have a weather dataset of 26 North American cities from roughly 1960 to 2015, downloaded from the Global Climatology Network. The dataset consists of the following monthly weather data for each city for each year: 1. average maximum temperature(0.1řC) 2. average minimum temperature(0.1řC) 3. average precipitation(0.1mm) 4. average snowfall(mm) 5. average snow depth(mm)

Using Python and relevant data science libraries, We plan to build and train an accurate machine learning algorithm to classify given unlabelled weather data into their corresponding North American cities. We will try a few different algorithms, validate them, and then choose the one with the best results.

2 Set-up

We need to import Python's data science libraries and read the data file.

```
In [8]: import numpy as np
        import pandas as pd

        data = pd.read_csv('monthly-data-labelled.csv')
        #This is what the data looks like
        data.head()
```

```
Out[8]:         city  year    tmax-01    tmax-02    tmax-03    tmax-04     tmax-05  \
        0  Anchorage  1960 -46.516129  -9.482759  -9.677419  52.400000  140.967742
        1  Anchorage  1961 -26.096774 -44.571429 -35.064516  58.200000  140.193548
        2  Anchorage  1962 -59.225806 -31.750000 -18.903226  69.366667  111.419355
        3  Anchorage  1963 -39.290323 -11.357143  -1.451613  41.700000  134.258065
        4  Anchorage  1964 -59.129032 -24.655172 -35.096774  45.866667   99.903226

              tmax-06     tmax-07     tmax-08  ...     snwd-03    snwd-04  snwd-05  \
        0  173.166667  180.225806  168.064516  ...  290.903226  44.066667      0.0
        1  169.633333  178.645161  161.806452  ...  113.096774   8.433333      0.0
        2  159.633333  187.451613  176.483871  ...  128.645161   5.866667      0.0
        3  146.200000  185.612903  182.129032  ...   60.645161  78.766667      0.0
        4  173.566667  182.516129  163.483871  ...  114.793103  53.266667      0.0
```

```
   snwd-06  snwd-07  snwd-08  snwd-09    snwd-10     snwd-11     snwd-12
0      0.0      0.0      0.0      0.0   0.000000   29.433333   77.612903
1      0.0      0.0      0.0      0.0  45.032258   98.366667  147.258065
2      0.0      0.0      0.0      0.0   0.000000   10.000000   46.548387
3      0.0      0.0      0.0      0.0   8.161290   34.466667   18.032258
4      0.0      0.0      0.0      0.0  15.516129  148.133333  345.870968

[5 rows x 62 columns]
```

From this dataframe, we need a two-dimensional array consisting of weather data values without the city label, and another array consisting of the corresponding city labels. Machine learning models are trained using this format of dataset.

```
In [2]: X = data.drop('city', 1).values #remove 'city' column
        y = data['city'].values
```

3 Training and Validating

We will try three types of machine learning models: Naive Bayesian, k-Nearest Neighbours, and Support Vector Machine. We will create pipelines to scale each feature's values to lie between 0 and 1 before computing the classifications.

For each model, we will divide the data into training and validation sets. This is how we will find out the accuracy of the trained model - by testing the model on the validation set, which will have never been shown to the model during training.

```
In [3]: from sklearn.model_selection import train_test_split
```

3.1 Naive Bayes (NB)

```
In [4]: from sklearn.pipeline import make_pipeline
        from sklearn.preprocessing import MinMaxScaler
        from sklearn.naive_bayes import GaussianNB

        n = 30 #number of iterations
        total = 0
        for i in range(n):
            X_train, X_valid, y_train, y_valid = train_test_split(X, y)
            bayes_model = make_pipeline(
                MinMaxScaler(),
                GaussianNB()
                )
            bayes_model.fit(X_train, y_train)
            total += bayes_model.score(X_valid, y_valid)

        print("bayes model score: " + str(total/n))

bayes model score: 0.6514942528735633
```

The bayes model has a score of roughly 65% accuracy, which is terrible. Why is this the case? NB algorithm assumes input variables to be independent, and also assumes input data to be distributed normally. None of these assumptions would hold true for weather data. For example, temperature is highly correlated with snowfall and snow depth. Moreoever, monthly temperatures throughout the decades have no reason to be normally distributed. This explains the model's low score.

3.2 k-Nearest Neighbours (kNN)

```
In [5]: from sklearn.neighbors import KNeighborsClassifier

        n_nbs = 5 #numer of neighbours to consider
        n = 30 #number of iterations
        total = 0
        for i in range(n):
            X_train, X_valid, y_train, y_valid = train_test_split(X, y)
            knn_model = make_pipeline(
                MinMaxScaler(),
                KNeighborsClassifier(n_neighbors=n_nbs)
                )
            knn_model.fit(X_train, y_train)
            total += knn_model.score(X_valid, y_valid)

        print("knn model score: " + str(total/n))
```

```
knn model score: 0.6647126436781612
```

The result is around 65%, which is as bad as the NB approach. One guess as to why this algorithm is not effective is because we have mapped the values of all the features to the same scale. Since kNN measures the Euclidian distance between data points in the feature space to determine the class of the prediction input, scaling as mentioned results in all the features contributing approximately the same weight to the distance calcuation. We have no knowledge of which features most significantly affect classification, and it could be the case that some features should be more heavily weighed than others.

3.3 Support Vector Machine (SVM)

```
In [6]: from sklearn.svm import SVC

        n = 30 #number of iterations
        total = 0
        for i in range(n):
            X_train, X_valid, y_train, y_valid = train_test_split(X, y)
            svc_model = make_pipeline(
                MinMaxScaler(),
                SVC(kernel='rbf', C=5, gamma='scale', decision_function_shape='ovr')
                )
```

```
    svc_model.fit(X_train, y_train)
    total += svc_model.score(X_valid, y_valid)

print("svc model score: " + str(svc_model.score(X_valid, y_valid)))
```

```
svc model score: 0.8137931034482758
```

Our SVM model's accuracy is around 80% - much better than the other two. SVM draws optimal hyperplanes between classes, which is more flexible and accurate than the methods of the other two machine learning algorithms.

4 Conclusion

We trained three machine learning models (Naive Bayes, k-Nearest Neighbours, and Support Vector Machine) using a relatively small decades-long weather dataset with 60 features (5 weather measurements * 12 months). The models were trained to predict cities from weather data, then were evaluated for their accuracy. The SVM model had the top accuracy of around 80%, while the other two models had an accuracy of around 66%.

Our SVM model is far from being practically useful. The model can be improved by adding more rows; cities like Denver has fewer data points than the rest of the cities. Perhaps Denver's (and other cities') missing weather information can be researched and added to the table. Another way to improve the model is to add more relevant features to our data. Adding wind velocity and atmospheric pressure, for example, will significantly improve all our models' accuracy.

```
In [ ]:
```