Hwk1

July 4, 2019

1 Machine Learning and predictive Analytics

1.1 Assignment 1

Name: Troy Zhongyi Zhang
Netid: zhongyiz@uchicago.edu

1.1.1 Part A: Data Cleaning & Exploratory Analysis

```
In [1]: import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
        df = pd.read_csv('bottle.csv')
        df.shape
```

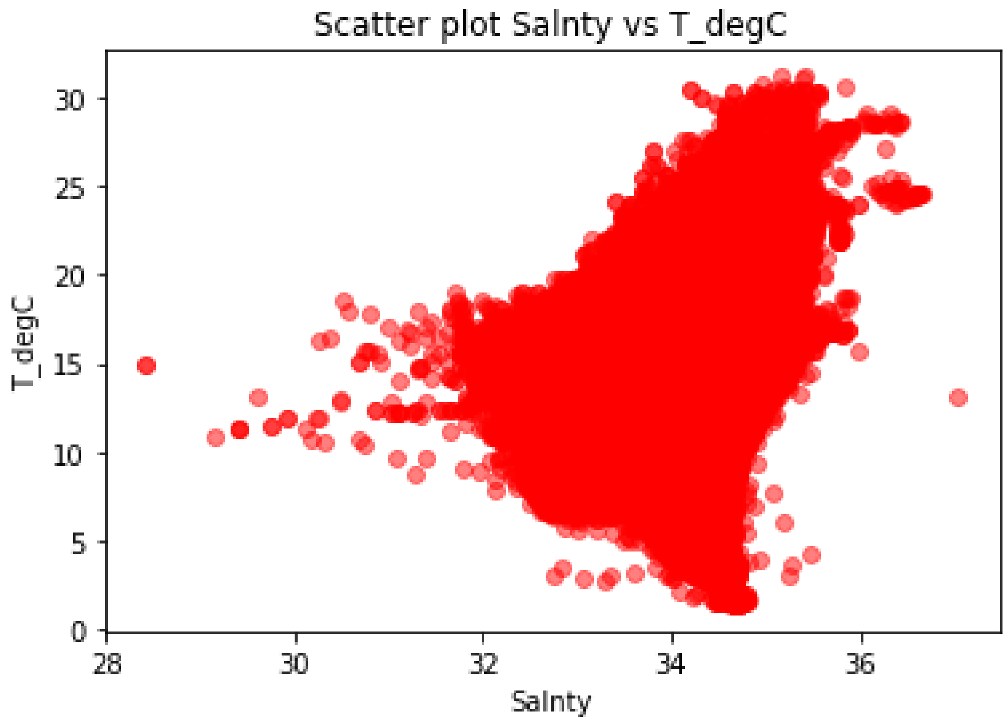
```
/Users/zhongyizhang/anaconda3/lib/python3.6/site-packages/IPython/core/interactiveshell.py:2785
  interactivity=interactivity, compiler=compiler, result=result)
```

```
Out[1]: (864863, 74)
```

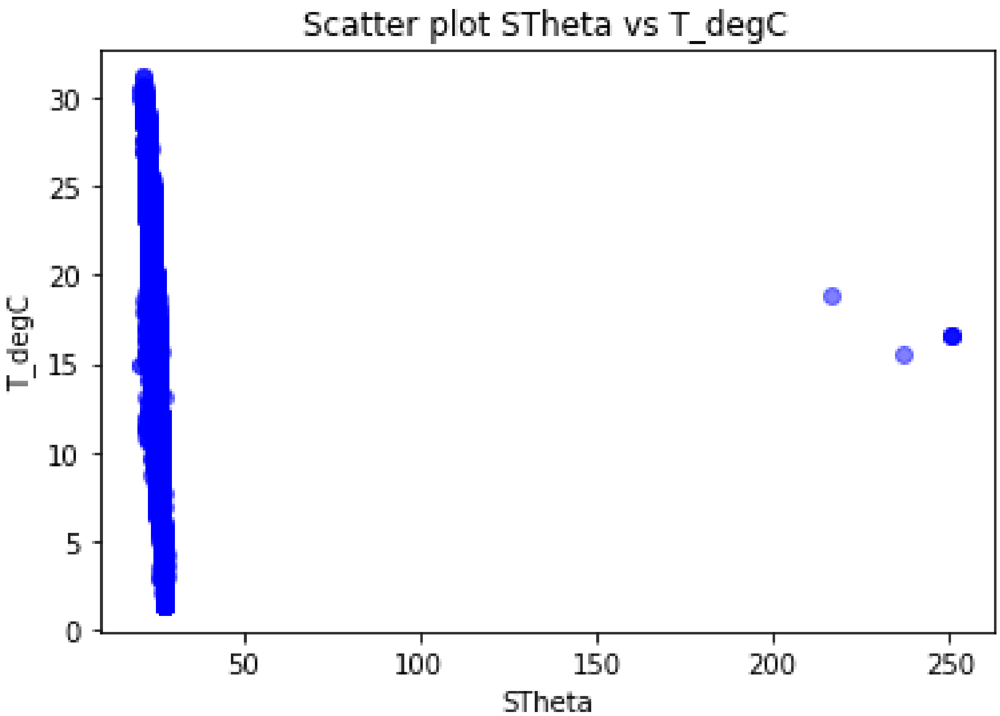
```
In [2]: df = df[['T_degC','Salnty','STheta']].dropna()
        df.shape
```

```
Out[2]: (812174, 3)
```

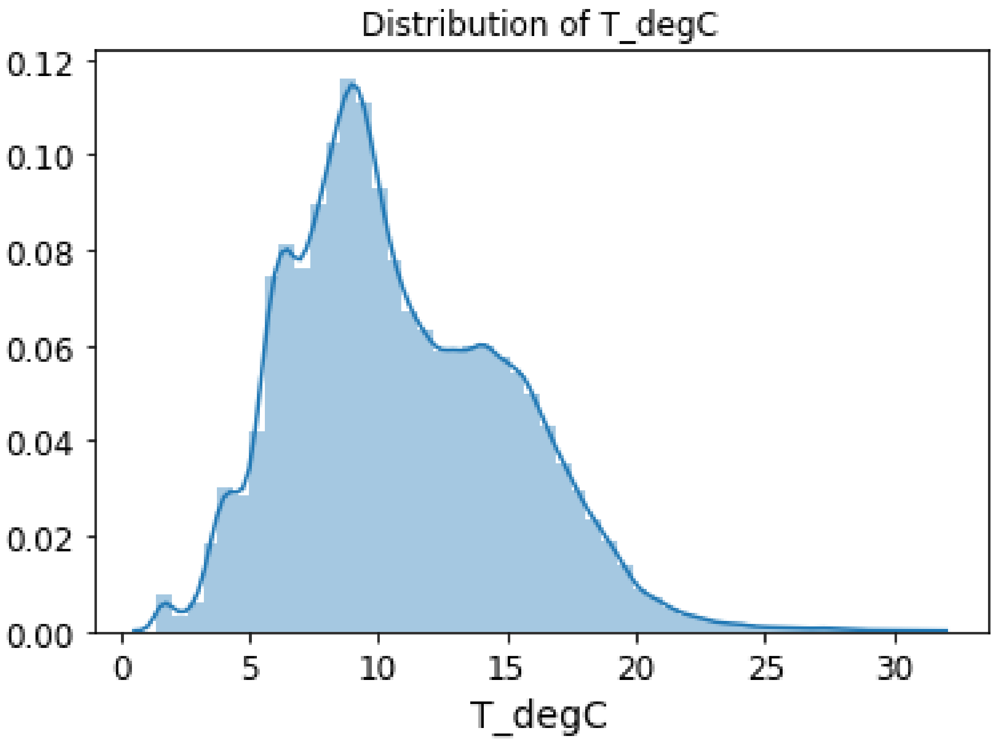
```
In [3]: x = df[['Salnty']]
        y = df[['T_degC']]
        plt.scatter(x, y, alpha=0.5, color = 'red')
        plt.title('Scatter plot Salnty vs T_degC')
        plt.xlabel('Salnty')
        plt.ylabel('T_degC')
        plt.show()
```




```
In [4]: x2 = df[['STheta']]
        y2 = df[['T_degC']]
        plt.scatter(x2, y2, alpha=0.5, color = 'blue')
        plt.title('Scatter plot STheta vs T_degC')
        plt.xlabel('STheta')
        plt.ylabel('T_degC')
        plt.show()
```



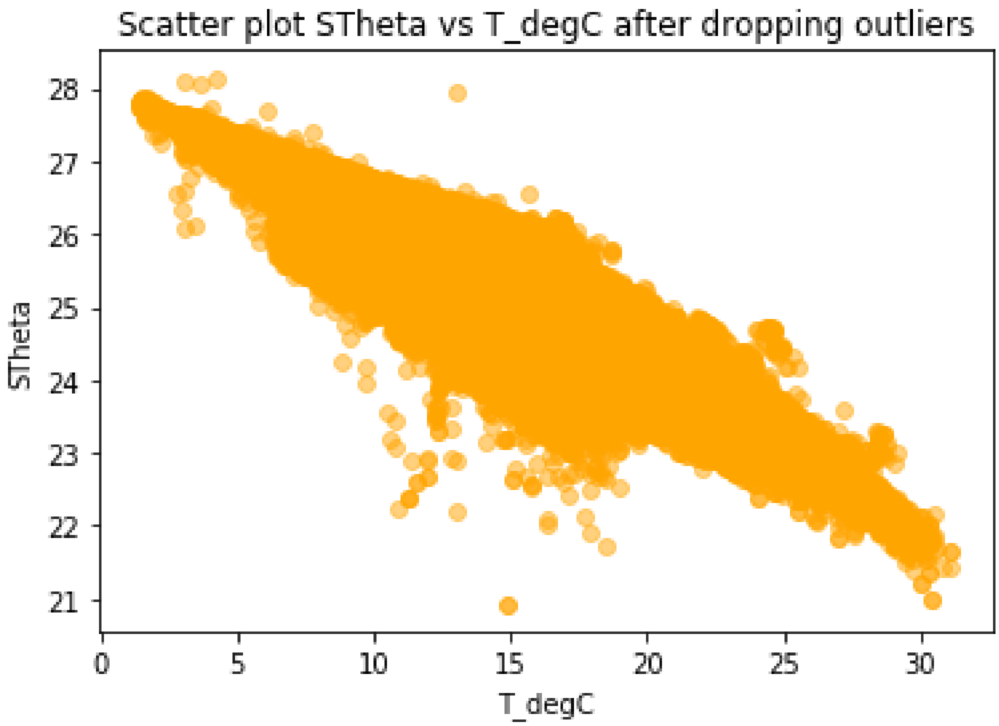

```
In [20]: ax = sns.distplot(df[['T_degC']])
         plt.title('Distribution of T_degC')
         plt.xlabel('T_degC')

/Users/zhongyizhang/anaconda3/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarn
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval

Out[20]: Text(0.5, 0, 'T_degC')
```




```
In [6]: df = df[df['STheta'] < 100]
        df.shape
```

```
Out[6]: (812168, 3)
```

```
In [7]: x3 = df[['T_degC']]
        y3 = df[['STheta']]
        plt.scatter(x3, y3, alpha=0.5, color = 'orange')
        plt.title('Scatter plot STheta vs T_degC after dropping outliers')
        plt.xlabel('T_degC')
        plt.ylabel('STheta')
        plt.show()
```



It did look better after dropping the outliers. Without dropping them, the plot cannot tell me any knowledge since all the points showed a straight vertical line. After dropping outliers, we can see an obivious descending trend for STheta as T_degC goes up.

1.1.2 Part B: Train and Test Split

```
In [8]: from sklearn.metrics import classification_report
        from sklearn.model_selection import cross_val_score
        import sklearn.model_selection as cv
        from sklearn.model_selection import train_test_split

        y = df.iloc[:,0:1]
        X = df.iloc[:,1:3]

        (X_train, X_test, y_train, y_test) = train_test_split(X, y, test_size=.20, random_state
```

1.1.3 Part C: Linear Regression Using Normal Equation - Coded In Python

```
In [9]: # To support both python 2 and python 3
        from __future__ import division, print_function, unicode_literals

        # Common imports
        import numpy as np
        import os
```

```
# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Where to save the figures
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "training_linear_models"

def save_fig(fig_id, tight_layout=True):
    path = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID, fig_id + ".png")
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format='png', dpi=300)

# Ignore useless warnings (see SciPy issue #5998)
import warnings
warnings.filterwarnings(action="ignore", message="^internal gelsd")
```

```
In [10]: X_b = np.c_[np.ones((649734, 1)), X_train]  # add x0 = 1 to each instance
         theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y_train)
```

```
In [11]: theta_best
```

```
Out[11]: array([[35.64451188],
                [ 3.11151204],
                [-5.03907257]])
```

```
In [12]: X_test.shape
```

```
Out[12]: (162434, 2)
```

```
In [13]: X_test_b = np.c_[np.ones((162434, 1)), X_test]  # add x0 = 1 to each instance
         y_pred = X_test_b.dot(theta_best)
         y_pred
```

```
Out[13]: array([[ 7.88437556],
                [ 7.14969722],
                [ 5.22668207],
                ...,
                [ 7.14752462],
                [16.11974133],
                [15.64019849]])
```

In [21]:
```
#mean-squared error
import sklearn.metrics as metrics
mse = metrics.mean_squared_error(y_test, y_pred)
print("Mean squared error:", mse)
```

```
Mean squared error: 0.23378301431781784
```

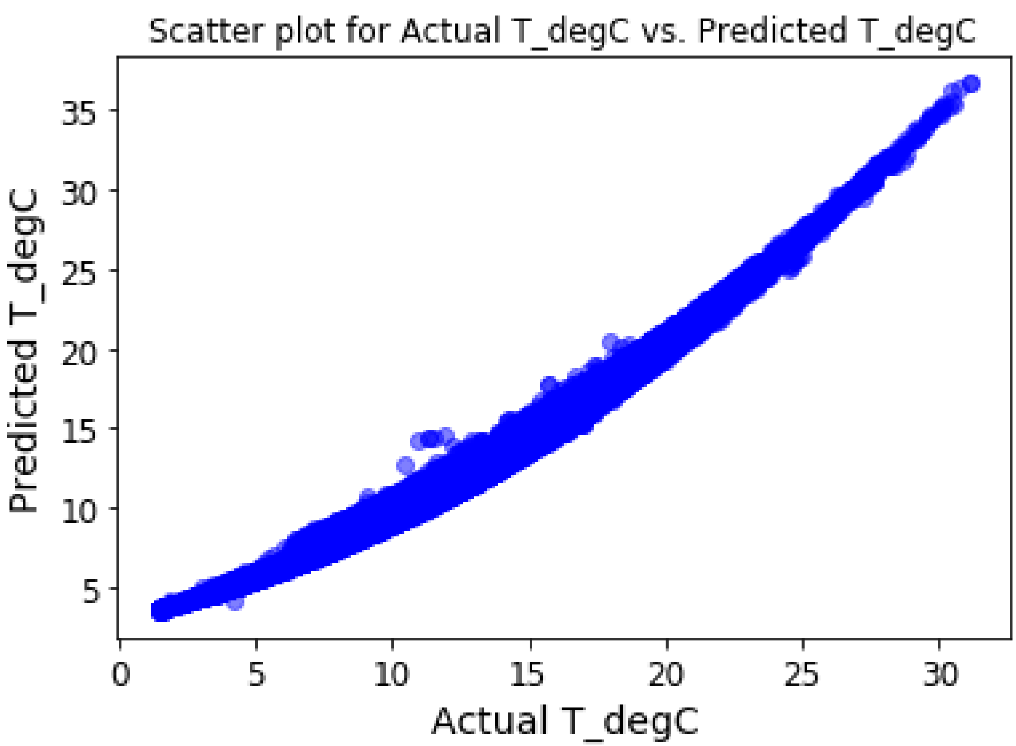
In [22]:
```
#r-squared
rs = metrics.r2_score(y_test, y_pred)
print("r-squared:", rs)
```

```
r-squared: 0.986891956563444
```

In [23]:
```
#Explained variance
ev = metrics.explained_variance_score(y_test, y_pred)
print("Explained variance:", ev)
```

```
Explained variance: 0.986891961456094
```

In [17]:
```
plt.scatter(y_test, y_pred, alpha=0.5, color = 'blue')
plt.title('Scatter plot for Actual T_degC vs. Predicted T_degC')
plt.xlabel('Actual T_degC')
plt.ylabel('Predicted T_degC')
plt.show()
```



1.1.4 Part D: Using sklearn API

In [24]:
```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)
lin_reg.intercept_, lin_reg.coef_
```

Out[24]:
```
(array([35.64451188]), array([[ 3.11151204, -5.03907257]]))
```

In [25]:
```
theta_best_svd, residuals, rank, s = np.linalg.lstsq(X_b, y_train, rcond=1e-6)
theta_best_svd
```

Out[25]:
```
array([[35.64451188],
       [ 3.11151204],
       [-5.03907257]])
```

In [26]:
```
#Alternatively
np.linalg.pinv(X_b).dot(y_train)
```

Out[26]:
```
array([[35.64451188],
       [ 3.11151204],
       [-5.03907257]])
```

In [27]:
```
y_pred_sklearn = lin_reg.predict(X_test)
y_pred_sklearn
```

Out[27]:
```
array([[ 7.88437556],
       [ 7.14969722],
       [ 5.22668207],
       ...,
       [ 7.14752462],
       [16.11974133],
       [15.64019849]])
```

The coefficients are exactly the same as what I found in Part C.

In [33]:
```
#mean-squared error
mse2 = metrics.mean_squared_error(y_test, y_pred_sklearn)
print("Mean squared error:", mse2)
```

```
Mean squared error: 0.2337830143181189
```

In [34]:
```
#r-squared
r2_2 = metrics.r2_score(y_test, y_pred_sklearn)
print("r-squared:", r2_2)
```

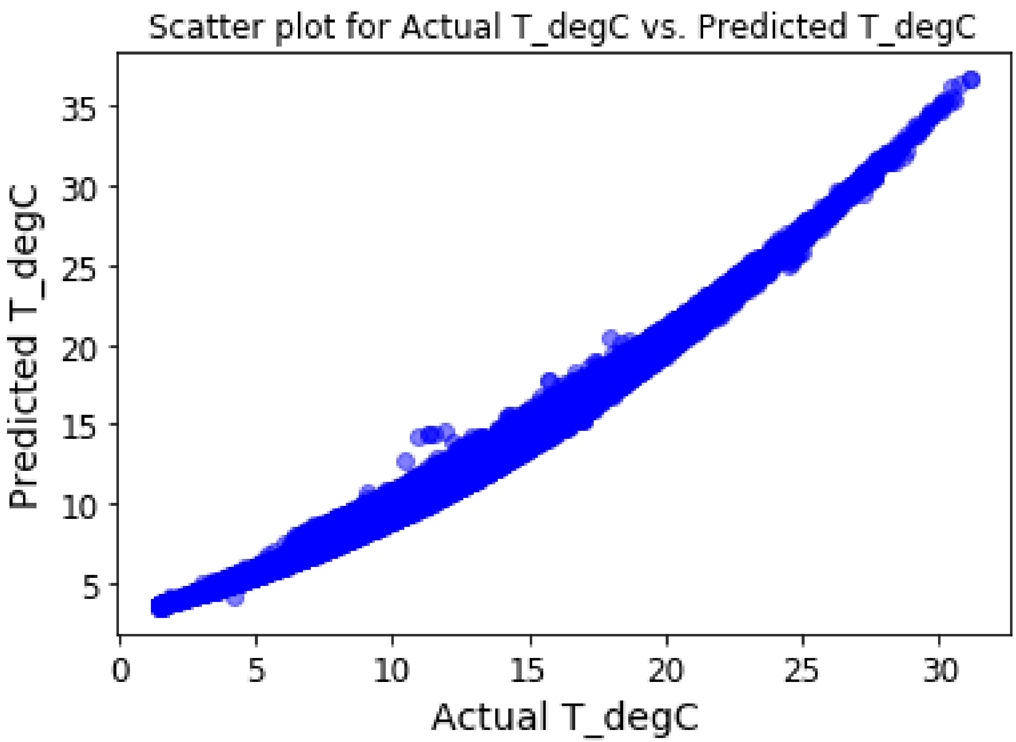
```
r-squared: 0.9868919565634271
```

```
In [35]: #Explained variance
         ev2 = metrics.explained_variance_score(y_test, y_pred_sklearn)
         print("Explained variance:", ev2)
```

```
Explained variance: 0.9868919614560767
```

```
In [36]: y_pred_sklearn = pd.DataFrame(y_pred_sklearn)
```

```
In [37]: #from pylab import *
         plt.scatter(y_test, y_pred_sklearn, alpha=0.5, color = 'blue')
         plt.title('Scatter plot for Actual T_degC vs. Predicted T_degC')
         plt.xlabel('Actual T_degC')
         plt.ylabel('Predicted T_degC')
         plt.show()
```



1.1.5 Part E: Conceptual Questions

1. Why is it important to have a test set?

2. If the normal equation always provides a solution, when would we not want to use it?

3. How might we improve the fit of our models from Part C & D?

- Note: There are lots of possible answers to this section - just describe one in detail.

4. As we move further into Machine Learning, we will need to continually consider the bias-variance tradeoff. Explain what bias is and what variance is in regards to the bias-variance tradeoff.
5. In a linear regression model, how might we reduce bias?

6. In a linear regression model, how might we reduce variance?

Answers:

1. The test set is used to measure the performance of the model. The test set allows me to compare different models in an unbiased way, by basing the comparisons in data that were not use in any part of the training/hyperparameter selection process.

2. If the condition number is small (one is the best possible), it doesn't matter much. However, if the condition number = 10ˆ9 with a stable method such as QR or SVD, I may have about 9 digits of accuracy in double precision. Forming the Normal equations, I've squared the condition number to 10ˆ18, and I will have essentially no accuracy in the answer.

3. We could increase more hyperparameters to improve the accuracies. Building a more complex model could capture the remaining variance to fit a better model. Adding interaction terms to model how two or more independent variables together impact the target variable. Another way is to add polynomial terms to model the nonlinear relationship between an independent variable and the target variable. Adding spines to approximate piecewise linear models. Fiting isotonic regression could remove any assumption of the target function form.

4. The bias–variance tradeoff is the property of a set of predictive models whereby models with a lower bias in parameter estimation have a higher variance of the parameter estimates across samples, and vice versa. The bias–variance dilemma or problem is the conflict in trying to simultaneously minimize these two sources of error that prevent supervised learning algorithms from generalizing beyond their training set.
   The bias is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).
   The variance is an error from sensitivity to small fluctuations in the training set. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).
   Tradeoff between bias and variance:
   – Simple Models: High Bias, Low Variance - underfitting
   – Complex Models: Low Bias, High Variance - overfitting

5. Choosing a representative training data set (making sure the training data is diverse and includes different groups is essential, but segmentation in the model can be problematic unless the real data is similarly segmented) and monitoring performance using real data can reduce bias.

6. If we want to reduce the amount of variance in a prediction, we must add bias. Increase training dataset size could reduce variance. Ensemble parameters from linear regression

could also reduce variance. Considering a linear regression model with three coefficients b0, b1, and b2, we could fit a group of linear regression models and calculate a final b0 as the average of b0 parameters in each model. Then we could repeat this process for b1 and b2.
