In [17]:

```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%config InlineBackend.figure_format = 'retina'
import math
```

In [18]:

```
import os
print(os.listdir("Dataset"))
```

```
['Data.zip']
```

In [19]:

```
df = pd.read_csv('Dataset/Data.zip')
df.head()
```

Out[19]:

	Date	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustDir	Win
0	2008-12-01	Albury	13.4	22.9	0.6	NaN	NaN	W	
1	2008-12-02	Albury	7.4	25.1	0.0	NaN	NaN	WNW	
2	2008-12-03	Albury	12.9	25.7	0.0	NaN	NaN	WSW	
3	2008-12-04	Albury	9.2	28.0	0.0	NaN	NaN	NE	
4	2008-12-05	Albury	17.5	32.3	1.0	NaN	NaN	W	

5 rows × 23 columns

In [20]:

```
df1 = df.drop(['Date','Location','WindGustSpeed','WindSpeed9am',
               'WindSpeed3pm','WindGustDir','WindDir9am','WindDir3pm'],
```

In [5]:

```
df1['RainToday'] = df1['RainToday'].map({'No' : 0, 'Yes' : 1})
```

In [6]:

```
df1['RainTomorrow'] = df1['RainTomorrow'].map({'No' : 0, 'Yes' : 1})
```

In [7]:

```
print(df1.MinTemp.mode())
print(df1.MaxTemp.mode())
print(df1.Rainfall.mode())
print(df1.Evaporation.mode())
print(df1.Sunshine.mode())
print(df1.Humidity9am.mode())
print(df1.Humidity3pm.mode())
print(df1.Pressure9am.mode())
print(df1.Pressure3pm.mode())
print(df1.Cloud9am.mode())
print(df1.Cloud3pm.mode())
print(df1.Temp9am.mode())
print(df1.Temp3pm.mode())
```

```
0    11.0
dtype: float64
0    20.0
dtype: float64
```

```
0    0.0
dtype: float64
0    4.0
dtype: float64
0    0.0
dtype: float64
0    99.0
dtype: float64
0    52.0
dtype: float64
0    1016.4
dtype: float64
0    1015.3
dtype: float64
0    7.0
dtype: float64
0    7.0
dtype: float64
0    17.0
dtype: float64
0    20.0
dtype: float64
```

In [8]:

```
df1.MinTemp.fillna(11.0, inplace=True)
df1.MaxTemp.fillna(20.0, inplace=True)
df1.Rainfall.fillna(0.0, inplace=True)
df1.Evaporation.fillna(4.0, inplace=True)
df1.Sunshine.fillna(0.0, inplace=True)
df1.Humidity9am.fillna(99.0, inplace=True)
df1.Humidity3pm.fillna(52.0, inplace=True)
df1.Pressure9am.fillna(1016.4, inplace=True)
df1.Pressure3pm.fillna(1015.3, inplace=True)
df1.Cloud9am.fillna(7.0, inplace=True)
df1.Cloud3pm.fillna(7.0, inplace=True)
df1.Temp9am.fillna(17.0, inplace=True)
df1.Temp3pm.fillna(20.0, inplace=True)
df1.RainToday.fillna(0, inplace=True)
df1.RainTomorrow.fillna(0, inplace=True)
```

In [9]:

```
df1.head().T
```

Out[9]:

	0	1	2	3	4
MinTemp	13.4	7.4	12.9	9.2	17.5
MaxTemp	22.9	25.1	25.7	28.0	32.3
Rainfall	0.6	0.0	0.0	0.0	1.0
Evaporation	4.0	4.0	4.0	4.0	4.0
Sunshine	0.0	0.0	0.0	0.0	0.0
Humidity9am	71.0	44.0	38.0	45.0	82.0
Humidity3pm	22.0	25.0	30.0	16.0	33.0
Pressure9am	1007.7	1010.6	1007.6	1017.6	1010.8
Pressure3pm	1007.1	1007.8	1008.7	1012.8	1006.0
Cloud9am	8.0	7.0	7.0	7.0	7.0
Cloud3pm	7.0	7.0	2.0	7.0	8.0
Temp9am	16.9	17.2	21.0	18.1	17.8
Temp3pm	21.8	24.3	23.2	26.5	29.7
RainToday	0.0	0.0	0.0	0.0	0.0

RainTomorrow	0.0	0.0	0.0	0.0	0.0

Маштабируем данные. Разделим выборку на обучающую и валидационную.

In [10]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df2 = df1.drop('RainTomorrow', axis=1)
X = scaler.fit_transform(df2)
from sklearn.model_selection import train_test_split

y = df1['RainTomorrow']

X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=0.
                                                        random_state=11)
```

In [11]:
```
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

In [12]:
```
from collections import Counter
from scipy.stats import mode
```

Реализация логистической регрессии

In [13]:
```
class MyLogisticRegression(BaseEstimator, ClassifierMixin):
    def __init__(self, lr = 0.1 , treshold = 0.5, epochs = 5000):
        self.lr = lr
        self.epochs = epochs
        self.treshold = treshold
        self.intercept = []
        self.x = []
        self.weight = []
        self.y = []

    #Сигмоида
    def sigmoid(self, x, weight):
        z = np.dot(x, weight)
        return 1 / (1 + np.exp(-z))

    #Функция потерь
    def loss(self, h, y):
        return (-y * np.log(h) - (1 - y) * np.log(1 - h)).mean()

    #Метод для подсчета гардиента
    def gradient_descent(self, X, h, y):
        return np.dot(X.T, (h - y)) / y.shape[0]


    def fit(self, x, y):
        self.intercept = np.ones((x.shape[0], 1))
        self.x = np.concatenate((self.intercept, x), axis=1)
        self.weight = np.zeros(self.x.shape[1])
        self.y = y
        for i in range(self.epochs):
            sigma = self.sigmoid(self.x, self.weight)

            loss = self.loss(sigma,self.y)

            dW = self.gradient_descent(self.x , sigma, self.y)
```

```
            #Обновляем веса
            self.weight -= self.lr * dW

        #return print('fitted successfully to data')

    #Method to predict the class label.
    def predict(self, x_new ):
        interc = np.ones((x_new.shape[0], 1))
        x_new = np.concatenate((interc, x_new), axis=1)
        result = self.sigmoid(x_new, self.weight)
        result = result >= self.treshold
        y_pred = np.zeros(result.shape[0])
        for i in range(len(y_pred)):
            if result[i] == True:
                y_pred[i] = 1
            else:
                continue

        return y_pred
```

Реализация метода ближайших соседей

In [15]:

```
class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self, k = 5):
        self.k = k
        self.x = []
        self.y = []


    def fit(self, x, y):
        self.x = x
        self.y = y

    def predict(self, x_test):
        Y_predict = np.zeros( len(x_test) )
        for i in range ( len(x_test) ) :
            x = x_test[i]
            neighbors = np.zeros( self.k )
            neighbors = self.find_neighbors( x )
            Y_predict[i] = mode( neighbors )[ 0 ][ 0 ]

        return Y_predict

    def euclidean( self , x, x_train ) :
        return np.sqrt(np.sum(np.square(x - x_train)))

    def find_neighbors( self , x ) :
        euclidean_distances = np.zeros(len(self.x))

        for i in range ( len(self.x) ) :
            d = self.euclidean( x, self.x[i] )
            euclidean_distances[i] = d

        inds = euclidean_distances.argsort()

        Y_train_sorted = []

        for i,y in enumerate(self.y):
                if i in inds[: self.k]:
                    Y_train_sorted.append(y)
        Y_train_sorted = np.array(Y_train_sorted)
```

```
        #return Y_train_sorted[: self.k]
        return Y_train_sorted
```

Реализация метода опорных векторов

In [21]:
```
class LinearSVM(BaseEstimator, ClassifierMixin):
    def __init__(self, C=1.0, lr=1e-3, epochs=500):
        self._support_vectors = None
        self.C = C
        self.beta = None
        self.b = None
        self.X = None
        self.y = None

        self.n = 0

        self.d = 0
        self.epochs = epochs
        self.lr = lr

    def __decision_function(self, X):
        return X.dot(self.beta) + self.b

    def __cost(self, margin):
        return (1 / 2) * self.beta.dot(self.beta) + self.C * np.sum(np.m

    def __margin(self, X, y):
        return y * self.__decision_function(X)

    def fit(self, X, y):
        self.n, self.d = X.shape
        self.beta = np.random.randn(self.d)
        self.b = 0

        self.X = X
        self.y = y

        self.y[y == 0] = -1

        loss_array = []
        for _ in range(self.epochs):
            margin = self.__margin(X, y)
            loss = self.__cost(margin)
            loss_array.append(loss)

            misclassified_pts_idx = np.where(margin < 1)[0]
            y1 = []
            for i,y in enumerate(self.y):
                if i in misclassified_pts_idx:
                    y1.append(y)
            y1 = np.array(y1)
            d_beta = self.beta - self.C * y1.dot(X[misclassified_pts_idx
            self.beta = self.beta - self.lr * d_beta

            d_b = - self.C * np.sum(y1)
            self.b = self.b - self.lr * d_b
            self._support_vectors = np.where(self.__margin(X, y) <= 1)[0

    def predict(self, X):
        return np.sign(self.__decision_function(X))
```

```
    def score(self, X, y):
        y[y == 0] = -1
        P = self.predict(X)
        return np.mean(y == P)
```

Реализация Naive Bayes

In [22]:

```
class NaiveBayes(BaseEstimator, ClassifierMixin):
    def __init__(self):
        self.X = []
        self.y = []
        self.info = {}

    def groupUnderClass(self, X_train, y_train):
        dict1 = {}
        for i,y in enumerate(y_train):
            if (y not in dict1):
                dict1[y] = []
            dict1[y].append(X_train[i])
        return dict1

    def mean(self, numbers):
        return sum(numbers) / float(len(numbers))

    def std_dev(self, numbers):
        avg = self.mean(numbers)
        variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(n
        return math.sqrt(variance)

    def MeanAndStdDev(self, X_train):
        info = [(self.mean(attribute), self.std_dev(attribute)) for attr
        return info

    def MeanAndStdDevForClass(self, X_train, y_train):
        info = {}
        dict1 = self.groupUnderClass(X_train, y_train)
        for classValue, instances in dict1.items():
            info[classValue] = self.MeanAndStdDev(instances)
        return info

    def fit(self, X, y):
        self.X = X
        self.y = y
        self.info = self.MeanAndStdDevForClass(self.X, self.y)


    def calculateGaussianProbability(self, x, mean, stdev):
        expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2
        return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo

    def calculateClassProbabilities(self, info, X_valid):
        probabilities = {}
        for classValue, classSummaries in info.items():
            probabilities[classValue] = 1
            for i in range(len(classSummaries)):
                mean, std_dev = classSummaries[i]
                x = X_valid[i]
                probabilities[classValue] *= self.calculateGaussianProbab
        return probabilities

    def forpredict(self, info, X_valid):
        probabilities = self.calculateClassProbabilities(info, X_valid)
```

```
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
            if bestLabel is None or probability > bestProb:
                bestProb = probability
                bestLabel = classValue
        return bestLabel


    # returns predictions for a set of examples
    def predict(self, X_valid):
        predictions = []
        for i in range(len(X_valid)):
            result = self.forpredict(self.info, X_valid[i])
            predictions.append(result)
        return predictions
```

In [23]:
```
from sklearn.model_selection import GridSearchCV
```

Настройка гиперпараметров моделей с помощью кросс валидации

In [18]:
```
parameters = {'lr': [0.001, 0.015, 0.01, 0.1, 0.15], 'treshold': [0.4, 0
logisticregression = MyLogisticRegression()
clf1 = GridSearchCV(logisticregression, parameters)
clf1.fit(X_train[: 5000], y_train[: 5000])
clf1.best_params_
```

Out[18]:
```
{'epochs': 1000, 'lr': 0.1, 'treshold': 0.5}
```

In [20]:
```
parameters = {'k':[5, 7, 10, 12, 15, 17, 20]}
knn = KNN()
clf2 = GridSearchCV(knn, parameters)
clf2.fit(X_train[: 5000], y_train[: 5000])
clf2.best_params_
```

Out[20]:
```
{'k': 5}
```

In [21]:
```
parameters = {'C':[1, 5, 10, 15, 20], 'epochs': [250, 500, 750, 1000]}
svm = LinearSVM()
clf3 = GridSearchCV(svm, parameters)
clf3.fit(X_train[: 5000], y_train[: 5000])
clf3.best_params_
```

Out[21]:
```
{'C': 1, 'epochs': 1000}
```

Обучаем модели и получаем оценки метрик

In [22]:
```
regressor = MyLogisticRegression(lr = 0.1, treshold = 0.5, epochs = 1000

regressor.fit(X_train[: 5000], y_train[: 5000])

predictions = []
predictions = regressor.predict(X_valid[: 1000])

accuracy = accuracy_score(y_valid[: 1000], predictions)
print("The accuracy of our classifier is {}".format(accuracy))

report = classification_report(y_valid[: 1000], regressor.predict(X_vali
print(report)
confusion_matrix(y_valid[: 1000], predictions)
```

```
The accuracy of our classifier is 0.835
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.96      0.90       776
         1.0       0.74      0.41      0.52       224

    accuracy                           0.83      1000
   macro avg       0.79      0.68      0.71      1000
weighted avg       0.82      0.83      0.82      1000
```

Out[22]:
```
array([[744,  32],
       [133,  91]], dtype=int64)
```

In [16]:
```python
knn = KNN()
knn.fit(X_train[: 5000], y_train[: 5000])
predictions = []
predictions = knn.predict(X_valid[: 1000])

accuracy = accuracy_score(y_valid[: 1000], predictions)
print("The accuracy of our classifier is {}".format(accuracy))

report = classification_report(y_valid[: 1000], predictions)
print(report)
confusion_matrix(y_valid[: 1000], predictions)
```

```
The accuracy of our classifier is 0.82
              precision    recall  f1-score   support

         0.0       0.85      0.93      0.89       776
         1.0       0.65      0.43      0.52       224

    accuracy                           0.82      1000
   macro avg       0.75      0.68      0.70      1000
weighted avg       0.80      0.82      0.81      1000
```

Out[16]:
```
array([[723,  53],
       [127,  97]], dtype=int64)
```

In [26]:
```python
svm = LinearSVM(C=1.0, epochs = 1000)
svm.fit(X_train[: 5000], y_train[: 5000])
print("train score:", svm.score(X_train[: 5000], y_train[: 5000]))
print("test score:", svm.score(X_valid[: 1000], y_valid[: 1000]))
y_valid1 = y_valid
y_valid1[y_valid1 == 0] = -1
predictions = []
predictions = svm.predict(X_valid[: 1000])
report = classification_report(y_valid1[: 1000], predictions)
print(report)
confusion_matrix(y_valid1[: 1000], svm.predict(X_valid[: 1000]))
```

```
train score: 0.7688
test score: 0.76
              precision    recall  f1-score   support

        -1.0       0.90      0.78      0.83       776
         1.0       0.48      0.70      0.57       224

    accuracy                           0.76      1000
   macro avg       0.69      0.74      0.70      1000
weighted avg       0.80      0.76      0.77      1000
```

Out[26]:
```
array([[603, 173],
       [ 67, 157]], dtype=int64)
```

In [26]:
```python
NB = NaiveBayes()
NB.fit(X_train[: 5000], y_train[: 5000])
```

```
predictions = []
predictions = NB.predict(X_valid[: 1000])

accuracy = accuracy_score(y_valid[: 1000], predictions)
print("The accuracy of our classifier is {}".format(accuracy))

report = classification_report(y_valid[: 1000], predictions)
print(report)
confusion_matrix(y_valid[: 1000], predictions)
```

```
The accuracy of our classifier is 0.73
              precision    recall  f1-score   support

        -1.0       0.90      0.73      0.81       776
         1.0       0.44      0.71      0.54       224

    accuracy                           0.73      1000
   macro avg       0.67      0.72      0.68      1000
weighted avg       0.80      0.73      0.75      1000
```

Out[26]:
```
array([[570, 206],
       [ 64, 160]], dtype=int64)
```

Коробочные решения

In [27]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
from sklearn.model_selection import cross_val_score

def LoRtrainer(X,y,final = False):
    print('Logistic Regression')
    from sklearn.linear_model import LogisticRegression
    classifier = LogisticRegression(max_iter=1000)
    classifier.fit(X,y)
    if final:
        return classifier
    else:
        accuracies = cross_val_score(estimator = classifier, X = X, y = 
        print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
        print("Standard Deviation: {:.2f} %".format(accuracies.std()*100
        print('')

def KNNtrainer(X,y,final = False):
    print('KNN Classifier')
    from sklearn.neighbors import KNeighborsClassifier
    classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkows
    classifier.fit(X,y)
    if final:
        return classifier
    else:
        accuracies = cross_val_score(estimator = classifier, X = X, y = 
        print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
        print("Standard Deviation: {:.2f} %".format(accuracies.std()*100
        print('')

def NBCtrainer(X,y,final = False):
    print('Naive Bayes Classifier')
    from sklearn.naive_bayes import GaussianNB
    classifier = GaussianNB()
    classifier.fit(X,y)
    if final:
        return classifier
    else:
        accuracies = cross_val_score(estimator = classifier, X = X, y = 
```

```
        print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
        print("Standard Deviation: {:.2f} %".format(accuracies.std()*100
        print('')

def SVCtrainer(X,y,final = False):
    print('SVM Classifier')
    from sklearn.svm import SVC
    classifier = SVC(kernel = 'linear')
    classifier.fit(X,y)
    if final:
         return classifier
    else:
        accuracies = cross_val_score(estimator = classifier, X = X, y =
        print("Accuracy: {:.2f} %".format(accuracies.mean()*100))
        print("Standard Deviation: {:.2f} %".format(accuracies.std()*100
        print('')



def TestEmAll(X,y):
    LoRtrainer(X,y)
    KNNtrainer(X[: 7000],y[: 7000])
    NBCtrainer(X,y)
    SVCtrainer(X[: 7000],y[: 7000])

TestEmAll(X_train, y_train)
```

```
Logistic Regression
Accuracy: 80.11 %
Standard Deviation: 0.16 %

KNN Classifier
Accuracy: 57.70 %
Standard Deviation: 1.23 %

Naive Bayes Classifier
Accuracy: 75.95 %
Standard Deviation: 0.25 %

SVM Classifier
Accuracy: 63.29 %
Standard Deviation: 1.11 %
```

In [33]:

```
import pickle
with open("my_models.pkl", "wb") as f:
    pickle.dump(clf1, f)
    pickle.dump(clf2, f)
    pickle.dump(clf3, f)
    pickle.dump(regressor, f)
    pickle.dump(knn, f)
    pickle.dump(svm, f)
    pickle.dump(NB, f)
```

Вывод

В результате проделанной лабораторной были реализованы методы логистической регрессии, SVM, KNN, Naive Bayes. По результатам проделанной реботы можно сделать вывод, что для данного набора данных лучше всего подходит логистическая регрессия которая дает точность около 83%. Следует заметить, что методы KNN и SVM обучаются оченьь долго на данной выборке, поэтому было уменьшено количество входных данных. Точность построенных моделей примерно совпадает с коробочными решениями.

```
In [ ]:
```