# Featuremap.py

In [1]:

```
import util
import numpy as np
import matplotlib.pyplot as plt

np.seterr(all='raise')

factor = 2.0


class LinearModel(object):
    """Base class for linear models."""

    def __init__(self, theta=None):
        """
        Args:
            theta: Weights vector for the model.
        """
        self.theta = theta

    def fit(self, X, y):
        """Run solver to fit linear model. You have to update the value of
        self.theta using the normal equations.

        Args:
            X: Training example inputs. Shape (n_examples, dim).
            y: Training example labels. Shape (n_examples,).
        """
        # *** START CODE HERE ***
        data=np.array(X)
        data.T
        xTx = data.T.dot(data)
        XtX = np.linalg.pinv(xTx)
        XtX_xT = XtX.dot(data.T)
        self.theta = XtX_xT.dot(y)
        return self.theta
        # *** END CODE HERE ***

    def fit_GD(self, X, y, alpha_value=0.01, iters=10000):
        """Run solver to fit linear model. You have to update the value of
        self.theta using the gradient descent algorithm.

        Args:
            X: Training example inputs. Shape (n_examples, dim).
            y: Training example labels. Shape (n_examples,).
        """
        # *** START CODE HERE ***
        counter = 0
        if self.theta == None:
            self.theta = [0] * np.shape(X)[1]
        while (counter < iters):
            for j in range(len(self.theta)):
                predict = (self.predict(X)-y) * X[:,j]
                para_delta = -alpha_value * (sum(predict))
                self.theta[j]=self.theta[j] + para_delta
            counter += 1
```

```
        # *** END CODE HERE ***

    def fit_SGD(self, X, y, alpha_value=0.01, iters=10000):
        """Run solver to fit linear model. You have to update the value of
        self.theta using the stochastic gradient descent algorithm.

        Args:
            X: Training example inputs. Shape (n_examples, dim).
            y: Training example labels. Shape (n_examples,).
        """
        # *** START CODE HERE ***
        counter=0
        if self.theta==None:
            self.theta = [0] * np.shape(X)[1]
        while (counter < iters):
            for i in range(len(X[:,0])):
                for j in range(len(self.theta)):
                    predict = (X[i].dot(self.theta) - y[i])
                    para_delta = alpha_value * predict * X[i][j]
                    self.theta[j] = self.theta[j] - para_delta
            counter += 1
        # *** END CODE HERE ***

    def create_poly(self, k, X):
        """
        Generates a polynomial feature map using the data x.
        The polynomial map should have powers from 0 to k
        Output should be a numpy array whose shape is (n_examples, k+1)

        Args:
            X: Training example inputs. Shape (n_examples, 2).
        """
        # *** START CODE HERE ***
        output_array = []
        for i in range ( len ( X [:,1] ) ):
            x = X[i][1]
            output_array.append([x**l for l in range(k+1)])
        return np.array(output_array)
        # *** END CODE HERE ***

    def create_cosine(self, k, X):
        """
        Generates a cosine with polynomial featuremap to the data x.
        Output should be a numpy array whose shape is (n_examples, k+2)

        Args:
            X: Training example inputs. Shape (n_examples, 2).
        """
        # *** START CODE HERE ***
        output_array=[]
        for i in range ( len ( X[:,1] ) ):
            x=X[i][1]
            poly=[x**l for l in range(k+1)]
            y = 1.5 * np.cos(13 * x)
            poly.append(y)
            output_array.append(poly)
        return (np.array(output_array))
        # *** END CODE HERE ***

    def predict(self, X):
        """
```

```
        Make a prediction given new inputs x.
        Returns the numpy array of the predictions.

        Args:
            X: Inputs of shape (n_examples, dim).

        Returns:
            Outputs of shape (n_examples,).
        """
        # *** START CODE HERE ***
        output_array = []
        for i in range ( len(X[:,0]) ):
            sum_predict=0
            for j in range(len(self.theta)):
                sum_predict += X[i][j] * self.theta[j]
            output_array.append(sum_predict)
        return output_array
        # *** END CODE HERE ***


    def predict_poly(self,X):
        output=[]
        #number of lines
        print(self.theta)

        for i in range (len(X)):
            sum_of_predictions=0
            for j in range(len(self.theta)):
                sum_of_predictions+=(X[i]**j)*self.theta[j]
            #print(sum_of_predictions)
            output.append(sum_of_predictions)
        return output

    def predict_cosine(self, X):
        output=[]
        #number of lines
        print(self.theta)

        for i in range (len(X)):
            sum_of_predictions=0

            for j in range(len(self.theta)):
                if j==len(self.theta)-1:
                    sum_of_predictions+=(np.cos(X[i]))*self.theta[j]
                else:
                    sum_of_predictions+=(X[i]**j)*self.theta[j]

            #print(sum_of_predictions)
            output.append(sum_of_predictions)

        return output

def run_exp(train_path, cosine=False, ks=[3, 5, 10, 20], filename='plot.pdf',fit_type='

    train_x, train_y = util.load_dataset(train_path, add_intercept=True)
    plot_x = np.ones([10000, 2])
    plot_x[:, 1] = np.linspace(-0.1, 1.1, 10000)
    plt.figure()
    plt.scatter(train_x[:, 1], train_y)

    for k in ks:
```

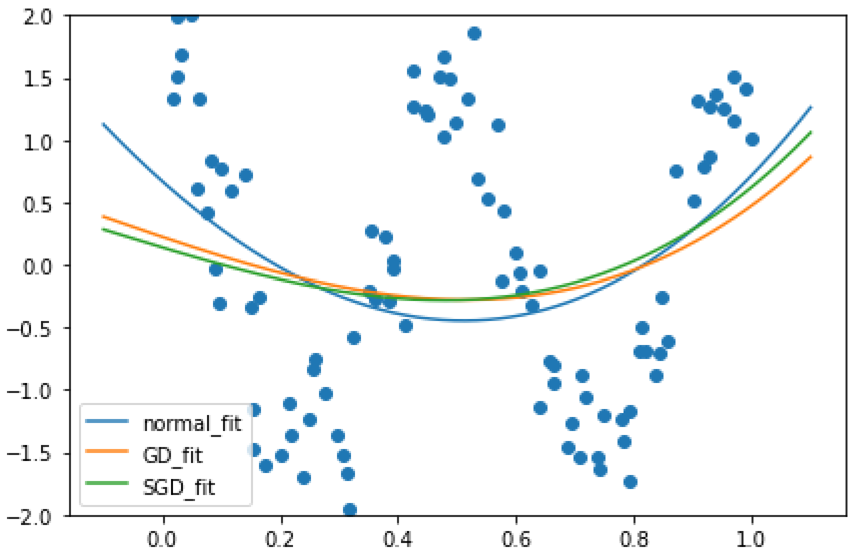
```
        '''
        Our objective is to train models and perform predictions on plot_x data
        '''
        # *** START CODE HERE ***
        if cosine==False:
            if fit_type == 'normal':
                model = LinearModel([0]*(k+1))
                training_data = model.create_poly(k,train_x)
                model.fit(training_data,train_y)
            if fit_type=='GD':
                model=LinearModel([0]*(k+1))
                training_data=model.create_poly(k,train_x)
                model.fit_GD(training_data,train_y)
            if fit_type=='SGD':
                model=LinearModel([0]*(k+1))
                training_data=model.create_poly(k,train_x)
                model.fit_SGD(training_data,train_y)
            plot_y = model.predict_poly(plot_x[:, 1])
        else:
            if fit_type=='normal':
                model=LinearModel([0]*(k+2))
                training_data=model.create_cosine(k,train_x)
                model.fit(training_data,train_y)
            if fit_type=='GD':
                model=LinearModel([0]*(k+2))
                training_data=model.create_cosine(k,train_x)
                model.fit_GD(training_data,train_y)
            if fit_type=='SGD':
                model=LinearModel([0]*(k+2))
                training_data=model.create_cosine(k,train_x)
                model.fit_SGD(training_data,train_y)
            plot_y = model.predict_cosine(plot_x[:, 1])
        # *** END CODE HERE ***
        '''
        Here plot_y are the predictions of the linear model on the plot_x data
        '''
        plt.ylim(-2.5, 2.5)
        plt.plot(plot_x[:, 1], plot_y, label='k=%d' % k)
    plt.legend()
    plt.tight_layout()
    plt.savefig(filename)
    plt.clf()
    return(plot_x[:,1],plot_y)

def main(medium_path, small_path):
    '''
    Run all expetriments
    '''
    # *** START CODE HERE ***
    # A1 Q1 Part 1.2
    model = LinearModel()
    train_x,train_y = util.load_dataset(medium_path,add_intercept=True)
    run_exp(medium_path,cosine=False,ks=[3],filename='1.2_degree-3_polynomial_regressio

    # A1 Q1 Part 1.3
    normal_model = LinearModel()
    gd_model = LinearModel()
    sgd_model = LinearModel()
    train_x,train_y = util.load_dataset(medium_path,add_intercept=True)
    plot_x = np.ones([1000, 2])
    plot_x[:, 1] = np.linspace(-factor * np.pi, factor * np.pi, 10000)
```

```
    ndx, ndy = run_exp(medium_path,cosine=False,ks=[3],fit_type='normal',output=False)
    gdx, gdy = run_exp(medium_path,cosine=False,ks=[3],fit_type='GD',output=False)
    sdx, sdy = run_exp(medium_path,cosine=False,ks=[3],fit_type='SGD',output=False)
    plt.scatter(train_x[:,1], train_y)
    plt.plot(ndx,ndy,label='normal_fit')
    plt.plot(gdx,gdy,label='GD_fit')
    plt.plot(sdx,sdy,label='SGD_fit')
    plt.ylim(-2,2)
    plt.legend()
    plt.savefig('1.3_degree-3_polynomial_GD_and_SGD.png')

    # A1 Q1 Part 1.4
    run_exp(medium_path, cosine=False,filename='1.4_degree-3_normal_fit_polynomial.png'
    run_exp(medium_path, cosine=False,filename='1.4_degree-3_GD_fit_polynomial.png',fit
    run_exp(medium_path, cosine=False,filename='1.4_degree-3_SGD_fit_polynomial.png',fi

    # A1 Q1 Part 1.5
    run_exp(medium_path, cosine=True,filename='1.5_other_feature_normal_fit_polynomial_
    run_exp(medium_path, cosine=True,filename='1.5_other_feature_GD_fit_polynomial_cosi
    run_exp(medium_path, cosine=True,filename='1.5_other_feature_SGD_fit_polynomial_cos

    # A1 Q1 Part 1.6
    run_exp(small_path, cosine=False,filename='1.6_overfitting_normal_fit_polynomial_co
    run_exp(small_path, cosine=False,filename='1.6_overfitting_GD_fit_polynomial_cosine
    run_exp(small_path, cosine=False,filename='1.6_overfitting_SGD_fit_polynomial_cosin
    # *** END CODE HERE ***


if __name__ == '__main__':
    main(medium_path='medium.csv',
         small_path='small.csv')
```

```
[ 0.66863478 -4.20104265  3.65396332  0.5956402 ]
[ 0.66863478 -4.20104265  3.65396332  0.5956402 ]
[0.22298266602110722, -1.583270646771859, 0.49525628347052175, 1.3395730176906706]
[0.13837886329031793, -1.4142313776445998, 0.4089405478281565, 1.4895580842433807]
[ 0.66863478 -4.20104265  3.65396332  0.5956402 ]
[   4.16053341  -77.0431555   372.31642346 -713.15312867  573.459009
 -157.20694037]
[ 1.49251638e+00  2.09679281e+01 -7.92353402e+02  8.10445549e+03
 -4.97752611e+04  1.93839114e+05 -4.67701633e+05  6.89533034e+05
 -6.03501132e+05  2.88115590e+05 -5.78432886e+04]
[ 1.18710238e+00  4.10817474e+01 -1.17241682e+03  1.07345301e+04
 -5.25297254e+04  1.40467283e+05 -1.70232071e+05 -4.99641909e+02
  1.41785885e+05  3.53659878e+04 -1.05578120e+05 -1.03551395e+05
  1.07448559e+04  1.06003520e+05  9.37536716e+04 -8.54536403e+03
 -1.04046995e+05 -9.50561635e+04  3.31005978e+04  1.45755342e+05
 -7.65410110e+04]
[0.22298266602110722, -1.583270646771859, 0.49525628347052175, 1.3395730176906706]
[0.347728158500078, -1.598138813276157, -0.22329844729901308, 0.27616652458823443, 0.815
2481127046745, 1.3633061891617293]
[0.2983034035384123, -0.9322482002843733, -0.36160407468979433, -0.43859276324028895, -
0.28173365207024414, 0.020531137863208065, 0.3347239103030575, 0.5950179404121966, 0.782
7123432345585, 0.9014615261524974, 0.9629920511883826]
[0.23414867177660248, -0.7554304353308451, -0.20055897068759604, -0.37791049541274424, -
0.3357393247846395, -0.13919508787712356, 0.08462671262622717, 0.2703239941468811, 0.397
6336467153324, 0.468145441266858, 0.49156954484688015, 0.4792008541808704, 0.44126682515
811566, 0.3861151542219343, 0.3201870195596438, 0.24827650198459147, 0.1738559038240080
3, 0.09937907236666886, 0.026534554021975024, -0.04355441477085757, -0.1101767284652051
5]
[0.13837886329031793, -1.4142313776445998, 0.4089405478281565, 1.4895580842433807]
[0.298587487153713, -1.4460543020914678, -0.38464419920279685, 0.23821195115289295, 0.89
0936953779157, 1.5272313947600635]
```

```
[0.2753385601166624, -0.8078826277799983, -0.4561171698734977, -0.4773071857598589, -0.2
9504352965705083, 0.015061265452158882, 0.33338998835750827, 0.5988054403417931, 0.79342
58945416807, 0.9205407341953659, 0.9912947874419699]
[0.2025323971014915, -0.6310003264536824, -0.2676164109346514, -0.38639417512943625, -0.
3273878837238857, -0.13578693661253716, 0.07843468812300422, 0.2559934637619823, 0.3780
311924878637, 0.44596811864560976, 0.46894746963906386, 0.4576905545509652, 0.4219513850
564636, 0.3697157007157835, 0.3071577312855601, 0.23887846618145755, 0.168212266013475
1, 0.09751544056397161, 0.02840833766310934, -0.038032779450754624, -0.1011284385415830
4]
[ 0.19608131 -2.42350574  5.3995528  -3.54315904  1.0056835 ]
[  0.25176118  -1.53103975 -11.97207875  63.93364114 -94.21597017
  43.57779905   0.96394965]
[-4.07306808e+00  2.47770220e+01 -3.56567302e+02  7.46106118e+03
 -4.44289245e+04  1.10729404e+05 -1.19011570e+05  1.76626850e+04
  7.24115505e+04 -5.84349998e+04  1.39426798e+04  3.66072578e+00]
[-1.69940636e+00  4.17783938e+01 -9.44623883e+02  1.08723689e+04
 -5.64021948e+04  1.39484266e+05 -1.38818095e+05 -3.70602625e+04
  1.22556994e+05  6.20702283e+04 -7.94887954e+04 -1.09930610e+05
 -1.34026488e+04  9.37354541e+04  1.03365101e+05  8.70980557e+03
 -9.98084353e+04 -1.07023488e+05  2.38675284e+04  1.57530685e+05
 -7.93549292e+04  1.91787299e+00]
[-0.010524833734595236, -0.15263380702710752, -0.00080023997353871, -0.0572484385631287
1, 1.0046354881980555]
[-0.018833116850751275, -0.14523030214302435, 0.040697603114526054, 0.00073886805279805
1, -0.05278306838897385, -0.08024473313231373, 1.0079158972698483]
[-0.0209479577737805, -0.1514929738540323, 0.05143018608527628, 0.021617930355179043, -
0.02668414120632385, -0.05178202374390915, -0.05421895236173626, -0.04208934231997192, -
0.022588328197717767, -0.0006062963187661011, 0.020930085248773538, 1.0099264047272798]
[-0.025971588967431628, -0.13272617222704336, 0.06694770943550973, 0.02604100776749484,
-0.0338043458884256, -0.06916893831400653, -0.08033219855553338, -0.07548962908402974, -
0.06201736141720626, -0.04498960993593182, -0.027499051014032126, -0.011256945691731201,
0.0029056446570902795, 0.014687626023111178, 0.02409062398131031, 0.03127739307081781,
0.03648616766409539, 0.0399803920724365, 0.042020395192741686, 0.04284846494337353, 0.04
2681962799055284, 1.0081922765312097]
[-0.019537564871237464, -0.1149762211787846, -0.02624234303731702, -0.0797333507113349,
1.0131741647473016]
[-0.02848247618340591, -0.1115740491225755, 0.017288620508573167, -0.010516114095918822,
-0.05527646028647103, -0.07998379942175549, 1.018201434892314]
[-0.02965465213471085, -0.11938767518117312, 0.023618006866022438, 0.007671590786770993,
-0.029465100954933755, -0.04967585758099512, -0.05055066443409117, -0.03807236661320493,
-0.018400986672246197, 0.003984286142364652, 0.026267231134310065, 1.0203092567480387]
[-0.034786048603750175, -0.09960084380473364, 0.04260878285472733, 0.014545623705466224,
-0.03613053442491025, -0.06867218798914734, -0.08014371260043152, -0.0765729114889136, -
0.06430843805235871, -0.04803890663221495, -0.030775054466812307, -0.014272162059755155,
0.0005531387392406881, 0.013309619451804014, 0.0239185621677106, 0.03247626822397659,
0.03916818546210305, 0.04421684062336381, 0.04785135893418401, 0.05029052191068292, 0.05
1734170039935454, 1.0165411523997858]
[  1.21042493 -23.75432314  71.82616617 -56.27566277]
[   1.13555388  -38.81949785  161.52046863 -162.46770609  -82.6448859
  130.17860501]
[  1.21874574 -32.21847769 101.67427887 -35.29281008 -66.18904719
 -41.67358187  -7.93706182  17.83993761  33.00396871  39.70240249
  40.8221525 ]
[  1.23092671 -31.23804395  94.03871495 -24.7995775  -57.5493064
 -42.22613091 -16.8542661    3.69205577  16.53919381  22.95989811
  25.01462016  24.45426315  22.51008224  19.96556374  17.28331974
  14.71579957  12.38559596  10.33890396   8.57933317   7.08850821
   5.8381995 ]
[0.16524803865020568, -0.3959172580379867, -0.22001975403706625, -0.19387716010305459]
[0.17722774532041125, -0.3684740845984443, -0.1945343109894537, -0.1720419521764463, -0.
15949197237724874, -0.14299668378165942]
[0.18304182122460502, -0.35149886118128615, -0.17774751508804804, -0.15711509979914723,
-0.14673637809583634, -0.13229444260109582, -0.11547125130579895, -0.0986303282361649, -
0.08307562048986447, -0.06933167803915175, -0.05750042396389289]
[0.18370838058029734, -0.34914963954116707, -0.17532432125077124, -0.1549061190218762, -
```

```
0.14481664349582024, -0.1306644070673145, -0.11410519326897349, -0.09749470382543665, -
0.08213654104045075, -0.06855792917673542, -0.056864517306465565, -0.046958857716845166,
-0.0386577503228417, -0.03175228760422807, -0.02603736104540874, -0.02132510857535317, -
0.017449974397546163, -0.01426947083535527, -0.011662843156042232, -0.00952882439973364
4, -0.007783112741985673 4]
[0.15983299499112028, -0.39224895382789043, -0.21993267497776875, -0.1935379370340887]
[0.17197398868859964, -0.3647967074822012, -0.19452831925875436, -0.17183553690219439, -
0.15900394245938693, -0.14238479542307128]
[0.17791454801249662, -0.3477823962400246, -0.17777419507670963, -0.156984992891165, -0.
14634148892638418, -0.131776969646754, -0.11493087400577158, -0.09811926431036908, -0.08
26169162393856, -0.06893245621745987, -0.0571596661476342]
[0.17860139897972835, -0.3454236051260991, -0.1753531773930222, -0.1547856956518572, -
0.1444345892662906, -0.13016036935835856, -0.113577529219959, -0.09699504944332969, -0.0
8168776331761878, -0.06816717672687198, -0.056530893172926734, -0.046677686568481896, -
0.03842288776644973, -0.031557331992055404, -0.025876255584933585, -0.02119240793549062
2, -0.01734093013649178, -0.014180022607922211, -0.011589564500601642, -0.00946885013348
6473, -0.007734062527911186]
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
```




```
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
<Figure size 432x288 with 0 Axes>
```

In [ ]: