# STATS 202: DATA MINING AND ANALYSIS HOMEWORK #3

INSTRUCTOR: LINH TRAN, HOMEWORK #3, DUE DATE: AUGUST 2, 2023, STANFORD UNIVERSITY, AND STUDENT: ADAM KAINIKARA

Problem 1 (7 points)

Chapter 6, Exercise 3 (p. 283).

$\Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2$ subject to $\Sigma_{j=1}^{p}|\beta| \leq s$

The left square part is RSS

a) $(iv)$ As we increase $s$ from 0 the training RSS will steadily increase. Increasing $s$ makes us restrict the $\beta$ (coefficients) less and less. By restricting coefficients less, the model will become more flexible. When the model becomes more flexible the training RSS will decrease.

b) $(ii)$ As we increase $s$ from 0 the test RSS would at first decrease but then slowly start increasing and would form a $U$ shape. Increasing s makes us restrict the $\beta$ (coefficients) less and less. By restricting coefficients less, the model will become more flexible. However at some point on test data, the RSS will once again increase.

c) $(iii)$ As we increase $s$ from 0, variance would steadily increase. Increasing $s$ makes us restrict the $\beta$ (coefficients) less and less. By restricting coefficients less, the model becomes more flexible. As flexibility increases, variance increases. This effect is like the bias variance trade off graph.

d) $(iv)$ As we increase $s$ from 0, squared bias would steadily decrease. Increasing $s$ makes us restrict the $\beta$ (coefficients) less and less. By restricting coefficients less, the model becomes more flexible. As flexibility increases, squared bias decreases. This effect is like the bias variance trade off graph.

e) $(v)$ As we increase $s$ from 0, the irreducible error would stay the same. Irreducible error is always there and does not come from the fitted model. So changing the flexibility of the model will have no impact.

Problem 2 (7 points)

Chapter 6, Exercise 4 (p. 284).

$\Sigma_{i=1}^{n}(y_i - \beta_0 - \Sigma_{j=1}^{p}\beta_j x_{ij})^2 + \lambda\Sigma_{j=1}^{p}\beta_j^2$

Ridge?

a) As we increase $\lambda$ from 0 the training RSS will increase. Increasing $\lambda$ increases the penalty term. With the penalty increasing and becoming more significant, the coefficients decrease. This leads to more error.

b) Initial, increasing l shrinks the coefficients. So as it initially increases RSS will decrease since over fitting is reduced. However, if we continue to increase l, the model may become too simple (due to smaller coefficients) and start to under fit, leading to an increase in the test RSS.

c) As we increase $\lambda$ from 0 the variance will increase. With the coefficients changing as $\lambda$ increases, the variability between how well the models can fit increases which increases the variance.

d) As we increases $\lambda$ the bias will decrease. As $\lambda$ increases, and the coefficients decrease, the model may become more simple. This leads to a decrease in bias.

e) Remain constant. The irreducible error represents the inherent noise in the data that cannot be reduced through modeling. As $\lambda$ increases, irreducible error i always there.

Problem 3 (7 points)

Chapter 6, Exercise 9 (p. 286). Don't do parts (e), (f), and (g).

a) Split the data into a 75% training and 25% test.

b) Fit a linear model. Got a MSE for linear model of 1503017.4360986822 and a r squared value of 0.9134874545115684

c) Fit a ridge model. Got a MSE of 1502973.7121541149 and a r squared value of 0.9134921934779143.

d) Fit a lasso model. Got a MSE of 35152969.33526452 and 9 non zero coefficients

Coding question

Problem 4 (7 points)

Chapter 7, Exercise 1 (p. 321).

a) We are given that $f(x) = \beta_o + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta^4 (x - \xi)^3 +$ where $(x - \xi)^3$ is its normal polynomial self if $x \geq \xi$ and is 0 otherwise

We are also given that $f_1(x) = a_1 + b_1 x + c_1 x^2 + d_1 x^3$

We want to find the polynomial and coefficients such that $f(x) = f_1(x)$

In part 'a' it is given that $x \leq \xi$ and because of this $f(x) = \beta_o + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$

So in order for $f(x) = f_1(x)$ to be true the coefficients are:

$a_1 = \beta_0, b_1 = \beta_1, c_1 = \beta_2, d_1 = \beta_3$

b) Now given $f_2(x) = a_2 + b_2 x + c_2 x^2 + d_2 x^3$

In part 'b' it is given that $x \geq \xi$ so $f(x) = \beta_o + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \xi)^3$

$= \beta_o + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x^3 - 3x^2\xi + 3x\xi^2 - \xi^3)$

Multiply out $\beta_4$ and rearrange the equation to follow the form of $f_2(x)\, ie: x^0, x^1, x^2 \ldots$

$(\beta_0 - \beta_4\xi^3) + (\beta_1 + 3\beta_4\xi^2)x + (\beta_2 - 3\beta_4\xi)x^2 + (\beta_3 + \beta_4)x^3$

In order for $f(x) = f_2(x)$

$a_2 = \beta_0 - \beta_4\xi^3, b_2 = \beta_1 + 3\beta_4\xi^2, c_2 = \beta_2 - 3\beta_4\xi, d_2 = \beta_3 + \beta_4$

c) Show$f_1(\xi) = f_2(\xi)$

$f_1(\xi) = \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3$

$f_2(\xi) = (\beta_0 - \beta_4\xi^3) + (\beta_1 + 3\beta_4\xi^2)\xi + (\beta_2 - 3\beta_4\xi)\xi^2 + (\beta_3 + \beta_4)\xi^3$

$\beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3 = (\beta_0 - \beta_4\xi^3) + (\beta_1 + 3\beta_4\xi^2)\xi + (\beta_2 - 3\beta_4\xi)\xi^2 + (\beta_3 + \beta_4)\xi^3$

$= \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3 = \beta_0 - \beta_4\xi^3 + \beta_1\xi + 3\beta_4\xi^3 + \beta_2\xi^2 - 3\beta_4\xi^3 + \beta_3\xi^3 + \beta_4\xi^3$

$\beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3 = \beta_0 + \beta_1\xi + \beta_2\xi^2 + \beta_3\xi^3$

d)$Show: f_1'(\xi) = f_2'(\xi)$

$f_1'(\xi) = \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2$

$f_2'(\xi) = -3\beta_4\xi^3 + \beta_1 + 9\beta_4\xi^2 + 2\beta_2\xi - 9\beta_4\xi^2 + 3\beta_3\xi^2 + 3\beta_4\xi^2$

$f_2'(\xi) = \beta_1 + 2\beta_2\xi + 3\beta_3\xi^2$

Therefore $f_1'(\xi) = f_2'(\xi)$

e)$Show: f_1''(\xi) = f_2''(\xi)$

$f_1''(\xi) = 2\beta_2 + 6\beta_3\xi$

$f_2''(\xi) = 2\beta_2 + 6\beta_3\xi$

Therefore $f_1''(\xi) = f_2''(\xi)$

Problem 5 (7 points)

Chapter 7, Exercise 8 (p. 324). Find at least one non-linear estimate which does better than linear regression, and justify this using a t-test or by showing an improvement in the cross-validation error with respect to a linear model. You must also produce a plot of the predictor X vs. the non-linear estimate fˆ(X).

Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=waylan to run on Wayland anyway. T-test statistic: 0.16492863090989973 P-value: 0.8690858275585926 Linear CV error: 6.502276260524977 Polynomial CV error: 6.502276260524972

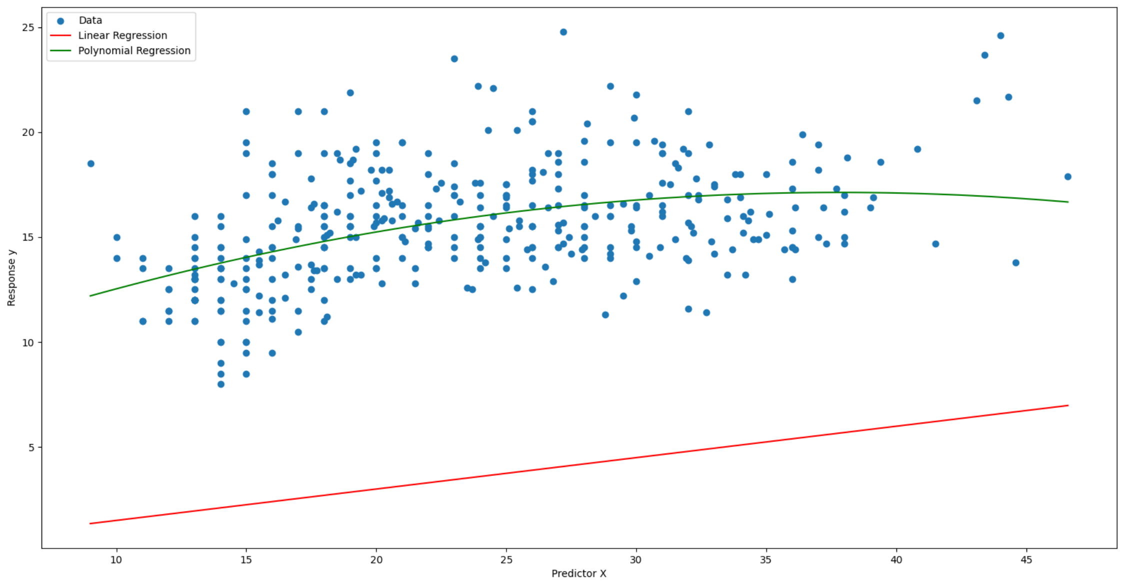


Figure 0.1. Poly and line

Problem 6 (7 points)
Chapter 9, Exercise 1 (p. 398).
Drawing a hyper plane

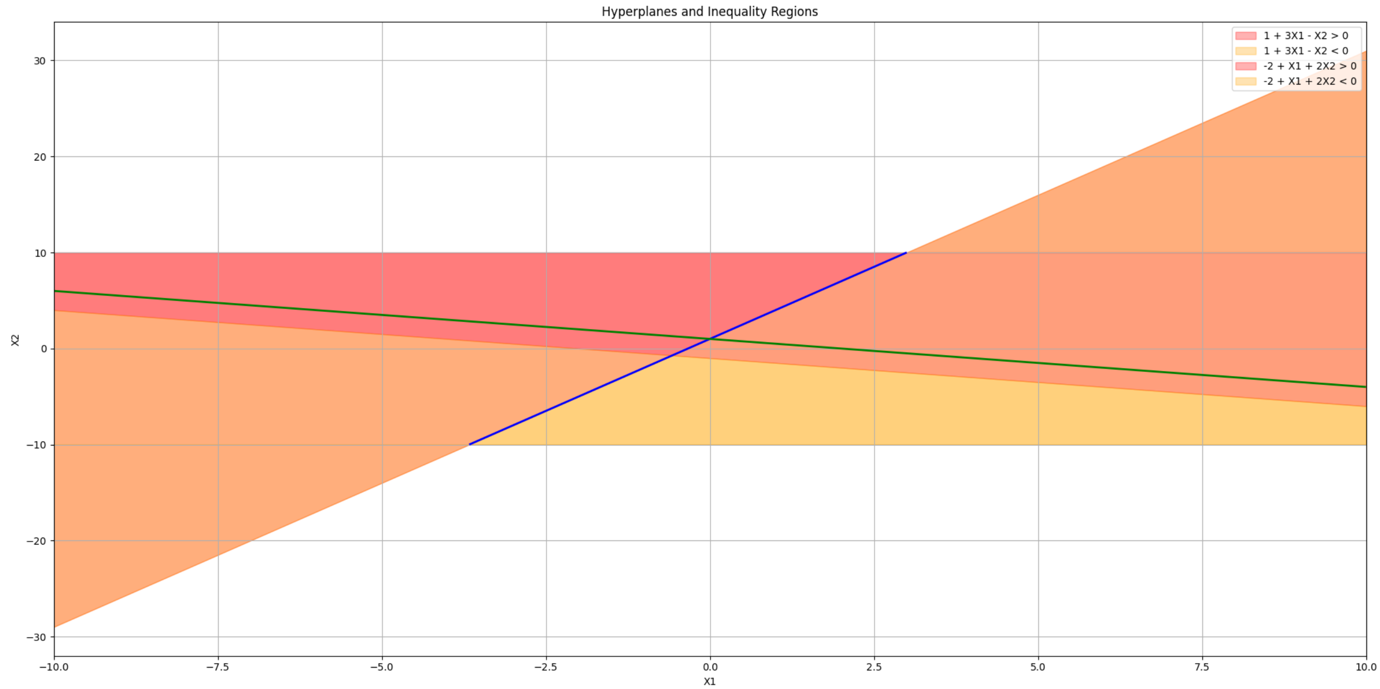


FIGURE 0.2. Hyper planes

Problem 7 (8 points)

Chapter 9, Exercise 8 (p. 401).

Note: Likely got some part of it wrong as I got the same accuracy scores for all and the best c value was 0.01.

Fitted a support vector classifier to the training data using C = 0.01, with Purchase as the response and the other variables as predictors. There were 612 support points. Training accuracy score of 0.38249999999999995 and test accuracy score of 0.4111111111111111 0.1 Fitted a support vector classifier to the training data using the best C = 0.1, and got Training accuracy score of 0.38249999999999995 and test accuracy score of 0.4111111111111111 Fitted a support vector classifier to the training data using C = 0.01, with Purchase as the response and the other variables as predictors. Used radial. There were 612 support points. Training accuracy score using radial is 0.38249999999999995 and test accuracy score of 0.4111111111111111 Fitted a support vector classifier to the training data using the best C = 0.01 with radial, and got Training accuracy score of 0.38249999999999995 and test accuracy score of 0.4111111111111111 Fitted a support vector classifier to the training data using C = 0.01, with Purchase as the response and the other variables as predictors. Used poly. There were 612 support points. Training accuracy score using radial is 0.38249999999999995 and test accuracy score of 0.4111111111111111 Fitted a support vector classifier to the training data using the best C = 0.01 with poly, and got Training accuracy score of 0.38249999999999995 and test accuracy score of 0.4111111111111111

```
 _             _____               _      
| |____      _|___ /  ___ ___   __| | ___ 
| '_ \ \ /\ / / |_ \ / __/ _ \ / _` |/ _ \
| | | \ V  V / ___) | (_| (_) | (_| |  __/
|_| |_|\_/\_/ |____/ \___\___/ \__,_|\___|
```

# ch6p9.py

```python
from wsgiref.headers import tspecials
from numpy import *
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LassoCV


# THIS IS CH6 P9 FOR QUESTION 3
def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18), delimiter=',')

    return data_a

def lin_model(x_a, y_v):
    #y_v = X@B
    b_v = linalg.pinv (x_a)@ y_v
    return b_v

def lin_fit(x_a, b_v):
    ypred_v = x_a@b_v
    return ypred_v

def rid_reg(xtrain, ytrain, xtest,ytest):
    alphas = logspace(-2, 2, 5)


    rid_model = RidgeCV(alphas=alphas, store_cv_values=True)

    rid_model.fit(xtrain, ytrain)
    ypred = rid_model.predict(xtest)
    rsq = 1 - (var(ytest-ypred))/(var(ytest))
    test_error = mean((ytest-ypred)**2)

    return rsq, test_error
def lass_reg(xtrain, ytrain, xtest,ytest):

    alphas = logspace(-2, 2, 5)
    lass_model = LassoCV(alphas=alphas)
    lass_model.fit(xtrain, ytrain)
    ypred = lass_model.predict(xtest)
    rsq = 1 - (var(ytest-ypred))/(var(ytest))
    test_error = mean((ytest-ypred)**2)
    non_zero= sum(lass_model.coef_ != 0)
    return rsq, test_error, non_zero


def main():
    data_a = data_loader("College.csv")
    #print(data_a)
    #print(len(data_a))
    '''Getting data set up'''
    n = int(0.75*len(data_a))

    xtrain = data_a[:n, 1:]
    xtrainreal = hstack((ones((xtrain.shape[0], 1)), xtrain))
    ytrain = data_a[:n ,:1]

    xtest = data_a[n:, 1:]
    xtestreal = hstack((ones((xtest.shape[0], 1)), xtest))
    ytest = data_a[n:, :1]

    print(xtrainreal.shape)
    print(ytrain.shape)
    print(xtestreal.shape)
    print(ytest.shape)

    '''Doing fit'''
    coef = lin_model(xtrainreal, ytrain)

    print(coef)

    pred_apps = lin_fit(xtestreal, coef)
    print(pred_apps)
    test_error = mean((ytest-pred_apps)**2)

    rsq = 1 - (var(ytest-pred_apps))/(var(ytest))
```

```python
    #print(rsq)
    print(f'The mse for lin reg is {test_error} and the r squared value for lin is {rsq}')

    rsqrid, mserid = rid_reg(xtrainreal, ytrain, xtestreal, ytest)
    print(f'The r squared value for ridge is {rsqrid} and the mse for ridge regression is {mserid}')

    rsqlass, mselass, nonzero = lass_reg(xtrainreal, ytrain, xtestreal, ytest)
    print(f'The r squared value for lasso is {rsqlass} and the mse for lasso  is {mselass} and {nonzero} coefficents')

if __name__ == '__main__':
    main()
```

# ch7p8.py

```
from numpy import *
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LassoCV
from scipy.stats import ttest_ind
from scipy.stats import t
from numpy import *
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy import stats
from scipy.interpolate import CubicSpline
from scipy.interpolate import splrep, BSpline
import statsmodels.api as sm

#THIS IS CH7 P8 FOR QUESTION 5

'''

To whoever is grading this problem: This file was a huge hot mess and still is a bit of a hot mess I'm sorry. I tried to clean it up a bit.
'''

def data_loader(fname):
    data_a = loadtxt(fname,skiprows=1, usecols=(1,2,3,4,5,6,7), delimiter=',')
    name_v = loadtxt(fname,skiprows = 1, usecols=(0), delimiter=',')
    return data_a, name_v


def lin_regression(x_a,y_v,name=''):
    if name:
        print(f'\n\n---------------------- {name} -------------------')
        model = sm.OLS(y_v, x_a)
        results = model.fit()
        print(results.summary())
    #Using stats models to get p value even though I did my own regression
    # y_v = X@B
    b_v = linalg.pinv(x_a)@y_v
    #print(b)
    return b_v

def fitted_func(x_a,b_v):
    yfit_v = x_a@b_v
    # with np.printoptions(precision=2):
    #     print(f'predicted mpg of cars {yfit_v=}')
    return yfit_v

def r_square(y_v,yfit_v):
    # This function is to find the r squared value
    # This will be calcualted by doing 1 - variance of (actual - predicited)/variance of actual
    rsq = 1 - (var(y_v-yfit_v))/(var(y_v))

    return rsq

def scatter_matrix(data_a, name_l):
    n, p = data_a.shape
    fig, axs = plt.subplots(4, 7)
    ax_l = list(axs.flat)
    mpl.rcParams['figure.autolayout'] = True
    font = {'family' : 'normal',
        'weight' : 'bold',
        'size'   : 10}

    #mpl.rc('font', **font)
    for i in range(p):
        for j in range(i+1,p):
            print(i, j, name_l[i], name_l[j])
            x_v = data_a[:,i]
            y_v = data_a[:,j]
            ax = ax_l.pop(0)
            ax.scatter(x_v, y_v, s=2**2)
            title = f'{name_l[i]} vs {name_l[j]}'
            ax.set_title(title[:25])
    plt.tight_layout()
    plt.show()

def fit_polynomial_regression(data_a, name_v, degree):
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(data_a.reshape(-1, 1))
    model = LinearRegression()
    model.fit(X_poly, name_v)
    return model

def fit_polynomial_regression(data_a, name_v, degree):
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(data_a.reshape(-1, 1))
def fit_polynomial_regression2(data_a, name_v, degree):
    poly = PolynomialFeatures(degree=degree)
    X_poly = poly.fit_transform(data_a.reshape(-1, 1))
    model = LinearRegression()
    model.fit(X_poly, name_v)
    return model

def lass_reg(xtrain, ytrain, xtest,ytest):

    alphas = logspace(-2, 2, 5)
    lass_model = LassoCV(alphas=alphas)
    lass_model.fit(xtrain, ytrain)
    ypred = lass_model.predict(xtest)
```

```python
    rsq = 1 - (var(ytest-ypred))/(var(ytest))
    test_error = mean((ytest-ypred)**2)
    non_zero= sum(lass_model.coef_ != 0)
    return rsq, test_error, non_zero


def fit_linear_regression(data_a, name_v):
    X_linear = column_stack((ones(len(data_a)), data_a))
    model = LinearRegression()
    model.fit(X_linear, name_v)
    return model

def fit_quadratic_regression(data_a, name_v):
    X_quad = column_stack((ones(len(data_a)), data_a, data_a**2))
    model = LinearRegression()
    model.fit(X_quad, name_v)
    return model

def perform_t_test(linear_model, quadratic_model, data_a, name_v):
    linear_residuals = name_v - linear_model.predict(column_stack((ones(len(data_a)), data_a)))
    quadratic_residuals = name_v - quadratic_model.predict(column_stack((ones(len(data_a)), data_a, data_a**2)))
    t_stat = (linear_residuals.T @ linear_residuals - quadratic_residuals.T @ quadratic_residuals) / len(name_v)
    return t_stat

def compute_cross_val_error(model, X, name_v):
    cv_error = mean(cross_val_score(model, X, name_v, scoring='neg_mean_squared_error', cv=5))
    return -cv_error

def plot_results(data_a, name_v, X_pred, linear_estimate, quadratic_estimate):
    plt.scatter(data_a, name_v, label='Data')
    plt.plot(X_pred, linear_estimate, label='Linear Regression', color='r')
    plt.plot(X_pred, quadratic_estimate, label='Quadratic Polynomial Regression', color='g')
    plt.xlabel('Predictor X')
    plt.ylabel('Response y')
    plt.legend()
    plt.show()

def main():
    data_a, name_v = data_loader('Auto.csv')

    name_v = name_v.astype(float)


    linear_model = fit_linear_regression(data_a, name_v)

    quadratic_model = fit_quadratic_regression(data_a, name_v)

    t_test_stat = perform_t_test(linear_model, quadratic_model, data_a, name_v)

    X = column_stack((data_a, data_a**2))  # Combine linear and quadratic features
    linear_cv_error = compute_cross_val_error(linear_model, X, name_v)
    quadratic_cv_error = compute_cross_val_error(quadratic_model, X, name_v)

    X_pred = linspace(data_a.min(), data_a.max(), 100)
    X_pred_reshaped = column_stack((ones(100), X_pred))  # Add a column of ones for linear regression
    linear_estimate = linear_model.predict(X_pred_reshaped)
    quadratic_estimate = quadratic_model.predict(column_stack((ones(100), X_pred, X_pred**2)))
    plot_results(data_a, name_v, X_pred, linear_estimate, quadratic_estimate)

    print("T-test statistic:", t_test_stat)
    print("Linear CV error:", linear_cv_error)
    print("Quadratic CV error:", quadratic_cv_error)
    polynomial_models = []
    for degree in range(1, 6):
        polynomial_model = fit_polynomial_regression(data_a, name_v, degree)
        polynomial_models.append(polynomial_model)

    linear_residuals = name_v - linear_model.predict(column_stack((ones(len(data_a)), data_a)))
    t_test_stat, p_value = ttest_ind(linear_residuals, zeros(len(linear_residuals)))  # Null hypothesis: linear model has no effect

    polynomial_t_stats = []
    polynomial_p_values = []
    for polynomial_model in polynomial_models:
        polynomial_residuals = name_v - polynomial_model.predict(PolynomialFeatures(degree=polynomial_model.degree).fit_transform(data_a.reshape(-1, 1)))
        t_stat, p_value = ttest_ind(polynomial_residuals, zeros(len(polynomial_residuals)))  # Null hypothesis: polynomial model has no effect
        polynomial_t_stats.append(t_stat)
        polynomial_p_values.append(p_value)

    #x_a, y_v, data_a, name_l  = data_loader('Auto.csv')
    #cor_a = corrcoef(data_a, rowvar=False)
    #print(cor_a.shape)
    #with np.printoptions(precision=4):
    #    print(cor_a)

    #print(x_a,y_v)
    #scatter_matrix(data_a, name_l)

    #print(x_a,y_v)
    #b_v = lin_regression(x_a,y_v, name='Main Regression')
    #yfit_v = fitted_func(x_a, b_v)
    #i_v = abs(b_v).argsort()[::-1]
    #print(f'Coefficients {b_v=}')


if __name__ == "__main__":
    main()
```

# ch9p1.py

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

# Hyperplane 1: 1 + 3*X1 - X2 = 0
# Hyperplane 2: -2 + X1 + 2*X2 = 0

#THIS IS CHAPTER 9 P 1 FOR QUESTION 6
x1 = np.linspace(-10, 10, 100)
x2 = np.linspace(-10, 10, 100)
X1, X2 = np.meshgrid(x1, x2)
hyperplane1 = 1 + 3 * X1 - X2
hyperplane2 = -2 + X1 + 2 * X2
plt.figure(figsize=(8, 6))
plt.contour(X1, X2, hyperplane1, levels=[0], colors='blue', linewidths=2)
plt.contour(X1, X2, hyperplane2, levels=[0], colors='green', linewidths=2)
plt.fill_between(x1, 1 + 3 * x1, 10, color='red', alpha=0.3, label='1 + 3X1 - X2 > 0')
plt.fill_between(x1, -10, 1 + 3 * x1, color='orange', alpha=0.3, label='1 + 3X1 - X2 < 0')
plt.fill_between(x1, (-2 - x1) / 2, 10, color='red', alpha=0.3, label='-2 + X1 + 2X2 > 0')
plt.fill_between(x1, -10, (-2 - x1) / 2, color='orange', alpha=0.3, label='-2 + X1 + 2X2 < 0')
plt.xlabel('X1')
plt.ylabel('X2')
plt.legend()
plt.title('Hyperplanes and Inequality Regions')
plt.grid(True)
plt.show()
```

# ch9p8.py

```python
from wsgiref.headers import tspecials
from numpy import *
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import RidgeCV
from sklearn.linear_model import LassoCV
from sklearn.linear_model import LassoCV
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

#THIS IS CH9 P 8 FOR QUESTION 7
def data_loader(fname):
    # 1WeekofPurchase 2StoreID  3PriceCH        4PriceMM        5DiscCH 6DiscMM 7SpecialCH      8SpecialMM      9LoyalCH        10SalePriceMM   11SalePriceCH   12PriceDiff
    #14PctDiscMM        15PctDiscCH     16ListPriceDiff 17STORE
    data_a = loadtxt(fname,skiprows=1, usecols=(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17), delimiter=',')
    #0 Purchase 13Store7
    #For store 7 i replaced no with 0 and yes with 1
    purchase_v = loadtxt(fname, skiprows=1, usecols=(0), delimiter=',', dtype=str)

    return data_a, purchase_v

def sup_vec_class(xtrain, ytrain):
    #https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
    #https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
    #https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/
    sup_vec_classifier = SVC(C=0.01)
    sup_vec_classifier.fit(xtrain, ytrain)
    support = len(sup_vec_classifier.support_vectors_)
    return sup_vec_classifier, support

def sup_vec_classv2(xtrain, ytrain,bestc):
    #https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
    #https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
    #https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/
    sup_vec_classifier = SVC(C=bestc)
    sup_vec_classifier.fit(xtrain, ytrain)
    #support = len(sup_vec_classifier.support_vectors_)
    return sup_vec_classifier

def sup_vec_class_rbf(xtrain, ytrain):
    sup_vec_class_radial = SVC(C=0.01, kernel='rbf')
    sup_vec_class_radial.fit(xtrain, ytrain)
    support = len(sup_vec_class_radial.support_vectors_)
    return sup_vec_class_radial, support
def sup_vec_class_rbfv2(xtrain, ytrain,bestcrad):
    #https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC
    #https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
    #https://pythonprogramming.net/linear-svc-example-scikit-learn-svm-python/
    sup_vec_class_best_c_radial = SVC(C=bestcrad, kernel='rbf')
    sup_vec_class_best_c_radial.fit(xtrain, ytrain)
    #support = len(sup_vec_classifier.support_vectors_)
    return sup_vec_class_best_c_radial

def sup_vec_class_poly(xtrain,ytrain):
    sup_vec_class_poly = SVC(C=0.01, kernel='poly', degree=2)
    sup_vec_class_poly.fit(xtrain,ytrain)
    support = len(sup_vec_class_poly.support_vectors_)
    return sup_vec_class_poly, support

def sup_vec_class_polyv2(xtrain, ytrain,bestpoly):
    sup_vec_class_best_c_poly = SVC(C=bestpoly, kernel='rbf')
    sup_vec_class_best_c_poly.fit(xtrain, ytrain)
    #support = len(sup_vec_classifier.support_vectors_)
    return sup_vec_class_best_c_poly

def acc_score(svmclass, xtrain,ytrain,xtest,ytest):
    ytrainpred = svmclass.predict(xtrain)
    trainscoretrain = 1 - accuracy_score(ytrain, ytrainpred)

    ytestpred = svmclass.predict(xtest)
    testscoretest = 1 - accuracy_score(ytest, ytestpred)
    return trainscoretrain, testscoretest

def find_best_c(svmclass,xtrain,ytrain):
    # this part was heavily influenced by
    #https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
    # https://scikit-learn.org/stable/modules/grid_search.html
    #https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/
    #https://stats.stackexchange.com/questions/305201/optimal-grid-search-for-c-in-svm
    #https://www.baeldung.com/cs/ml-svm-c-parameter
    #finding_c = logspace(-2, 1, 5) #5 values from 0.01 to 10
    param_grid = {'C': [ 0.1, 10]}
    grid_search = GridSearchCV(svmclass, param_grid, cv=100)
    grid_search.fit(xtrain, ytrain)
    best_c = grid_search.best_params_['C']
    return best_c
def find_best_c_radial(svmclassradial, xtrain, ytrain):
    #same as above but now with rbf
    param_grid_radial = {'C': [0.01, 10]}
    svm_cv_radial = SVC(kernel='rbf')
    grid_search_radial = GridSearchCV(svm_cv_radial, param_grid_radial, cv=100)
    grid_search_radial.fit(xtrain, ytrain)
    best_c_radial = grid_search_radial.best_params_['C']
    return best_c_radial

def find_best_c_poly(svmclasspoly, xtrain,ytrain):
    param_grid_poly = {'C': [0.01, 10]}
    svm_cv_poly = SVC(kernel='poly', degree=2)
    grid_search_poly = GridSearchCV(svm_cv_poly, param_grid_poly, cv=100)
    grid_search_poly.fit(xtrain, ytrain)
    best_c_poly = grid_search_poly.best_params_['C']
    return best_c_poly

def main():
    data_a, purchase_v  = data_loader('OJ.csv')


    xtrain, xtest, ytrain, ytest = train_test_split(data_a, purchase_v, train_size=800, random_state=42)
    #print(f'xtrain{xtrain} ytrain{ytrain} xtest{xtest}ytest{ytest}')

    '''part b'''
    svmclass, support = sup_vec_class(xtrain,ytrain)

    print(f'Fitted a support vector classifier to the training data using C = 0.01, with Purchase as the response and the other variablesas predictors. There were {support} support points.')

    '''part c'''
    trainscore, testscore = acc_score(svmclass, xtrain,ytrain,xtest,ytest)
    print(f'Training accuracy score of {trainscore} and test accuracy score of {testscore}')
    '''part d'''
    c = find_best_c(svmclass,xtrain,ytrain)
    print(c)
    '''part e'''
    svmclassv2 = sup_vec_classv2(xtrain,ytrain,c)
    newtrainscore, newtestscore = acc_score(svmclassv2, xtrain,ytrain,xtest,ytest)
    print(f'Fitted a support vector classifier to the training data using the best C = {c}, and got Training accuracy score of {newtrainscore} and test accuracy score of {newtestscore}')
```

```python

    '''now doing with radial'''
    svmclassradial, supportradial = sup_vec_class_rbf(xtrain, ytrain)
    print(f'Fitted a support vector classifier to the training data using C = 0.01, with Purchase as the response and the other variablesas predictors. Used radial. There were {supportradial
    trainscoreradial, testscoreradial = acc_score(svmclassradial, xtrain, ytrain, xtest, ytest)
    print(f'Training accuracy score using radial is {trainscoreradial} and test accuracy score of {testscoreradial}')
    bestcrad = find_best_c_radial(svmclassradial,xtrain,ytrain)
    svmclassradialv2 = sup_vec_class_rbfv2(xtrain, ytrain, bestcrad)
    newtrainscorerad, newtestscorerad = acc_score(svmclassradialv2,xtrain,ytrain,xtest,ytest)
    print(f'Fitted a support vector classifier to the training data using the best C = {bestcrad} with radial, and got Training accuracy score of {newtrainscorerad} and test accuracy score c

    '''now doing with poly'''
    svmclasspoly, supportpoly = sup_vec_class_poly(xtrain, ytrain)
    print(f'Fitted a support vector classifier to the training data using C = 0.01, with Purchase as the response and the other variablesas predictors. Used poly. There were {supportpoly} su
    trainscorepoly, testscorepoly = acc_score(svmclasspoly, xtrain, ytrain, xtest, ytest)
    print(f'Training accuracy score using radial is {trainscorepoly} and test accuracy score of {testscorepoly}')
    bestcpoly = find_best_c_poly(svmclasspoly,xtrain,ytrain)
    svmclasspolyv2 = sup_vec_class_polyv2(xtrain,ytrain,bestcpoly)
    newtrainscorepoly, newtestscorepoly = acc_score(svmclasspolyv2,xtrain,ytrain,xtest,ytest)
    print(f'Fitted a support vector classifier to the training data using the best C = {bestcpoly} with poly, and got Training accuracy score of {newtrainscorepoly} and test accuracy score c

if __name__ == '__main__':
    main()
```