Homework 7 **ISYE 6740**

Shrikanth Mahale (smahale6)

1. Random Forest for Email Spam Classifier

Your task for this question is to build a spam classifier using the UCR email spma dataset `https://archive.ics.uci.edu/ml/datasets/Spambase` came from the postmaster and individuals who had filed spam. The collection of non-spam e-mails came from filed work and personal e-mails, and hence the word 'george' and the area code '650' are indicators of non-spam. These are useful when constructing a personalized spam filter.

One would either have to blind such non-spam indicators or get a very wide collection of non-spam to generate a general purpose spam filter. Load the data.

Question 1.1

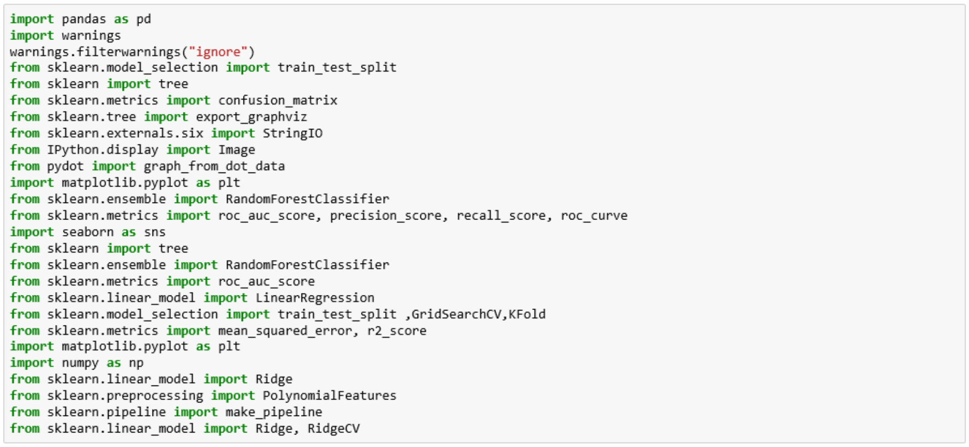
1. (5 points) How many instances of spam versus regular emails are there in the data? How many data points there are? How many features there are?

   Note: there may be some missing values, you can just fill in zero.

Solution 1.1

The steps given below were performed in order to solve these question

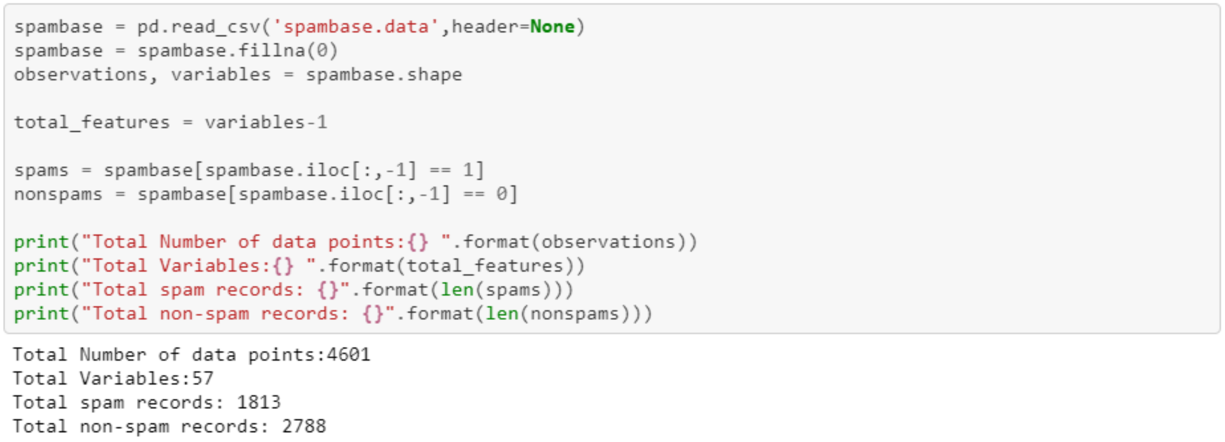
1) All the important modules pertaining to this HW was imported as shown in screenshot below



```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
from pydot import graph_from_dot_data
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, precision_score, recall_score, roc_curve
import seaborn as sns
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split ,GridSearchCV,KFold
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Ridge, RidgeCV
```

2) The dataset named '**Spambase.data**'was downloaded from the link given below
   http://archive.ics.uci.edu/ml/machine-learning-databases/spambase/

3) The dataset was imported into a pandas dataframe and the calculations requested in this question was performed as shown in the screenshot below



```
spambase = pd.read_csv('spambase.data',header=None)
spambase = spambase.fillna(0)
observations, variables = spambase.shape

total_features = variables-1

spams = spambase[spambase.iloc[:,-1] == 1]
nonspams = spambase[spambase.iloc[:,-1] == 0]

print("Total Number of data points:{} ".format(observations))
print("Total Variables:{} ".format(total_features))
print("Total spam records: {}".format(len(spams)))
print("Total non-spam records: {}".format(len(nonspams)))
```

```
Total Number of data points:4601
Total Variables:57
Total spam records: 1813
Total non-spam records: 2788
```

4) Given below are the observations/output

   Total Number of data points: **4601**
   Total Variables: **57**
   Total spam records: **1813**
   Total non-spam records: **2788**

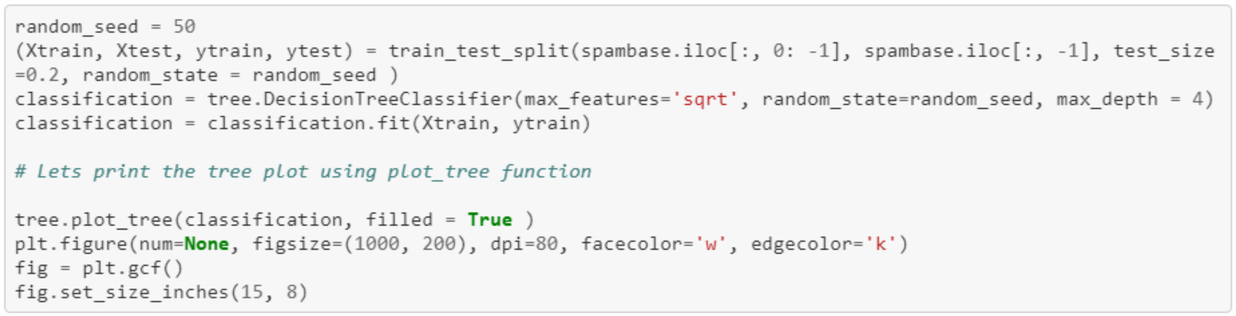
Question 1.2

2. (10 points) Build a classification tree model (also known as the CART model). In Python, this can be done using sklearn.tree.DecisionTreeClassifier. In our answer, you should report the tree models fitted similar to what is shown in the "Random forest" lecture, Page 16, the tree plot. In Python, getting this plot can be done using sklearn.tree.plot_tree function.

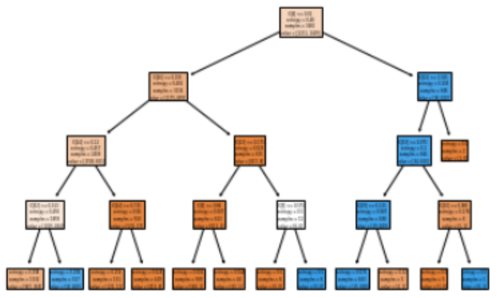
Solution 1.2

The following steps were performed to solve this question

1) The **Spambase.data** was shuffled randomly to split into test dataset and training dataset with 80% of data allotted to the training dataset and the remaining 20% for the test dataset.
2) A classification tree model was built using Scikit Learn's DecisionTreeClassifier module.
3) The Decision Tree was plotted using Scikit Learn's tree.plot tree module and the output is given in the screenshot below.



```
random_seed = 50
(Xtrain, Xtest, ytrain, ytest) = train_test_split(spambase.iloc[:, 0: -1], spambase.iloc[:, -1], test_size
=0.2, random_state = random_seed )
classification = tree.DecisionTreeClassifier(max_features='sqrt', random_state=random_seed, max_depth = 4)
classification = classification.fit(Xtrain, ytrain)

# Lets print the tree plot using plot_tree function

tree.plot_tree(classification, filled = True )
plt.figure(num=None, figsize=(1000, 200), dpi=80, facecolor='w', edgecolor='k')
fig = plt.gcf()
fig.set_size_inches(15, 8)
```



Question 1.3

3. (15 points) Also build a random forrest model. In Python, this can be done using sklearn.ensemble.RandomForestClassifier.

   Now partition the data to use the first 80% for training and the remaining 20% for testing. Your task is to compare and report the AUC for your classification tree and random forest models on testing data, respectively. To report your results, please try different tree sizes. Plot the curve of AUC versus Tree Size, similar to Page 15 of the Lecture Slides on "Random Forest".

   Background information: In classification problem, we use AUC (Area Under The Curve) as a performance measure. It is one of the most important evaluation metrics for checking any classification model?s performance. ROC (Receiver Operating Characteristics) curve measures classification accuracy at various thresholds settings. AUC measures the total area under the ROC curve. Higher the AUC, better the model is at distinguishing the two classes. If you want to read a bit more about AUC curve, check out this link `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5` For instance, in Python, this can be done using sklearn.metrics.roc_auc_score and you will have to figure out the details.

Solution 1.3

The following steps were performed to solve this question

1) A Random Forest Classifier was fitted on the training set generated in previous question and tested on the test set.
2) AUC was then calculated for Random Forest classifier and the classification tree classifier as shown in the screenshot below.
3) Following output was observed

   Decision Tree : Test ROC AUC Score: **0.9177935190036678**

   Random Forest: Test ROC AUC Score: **0.9870168473660876**

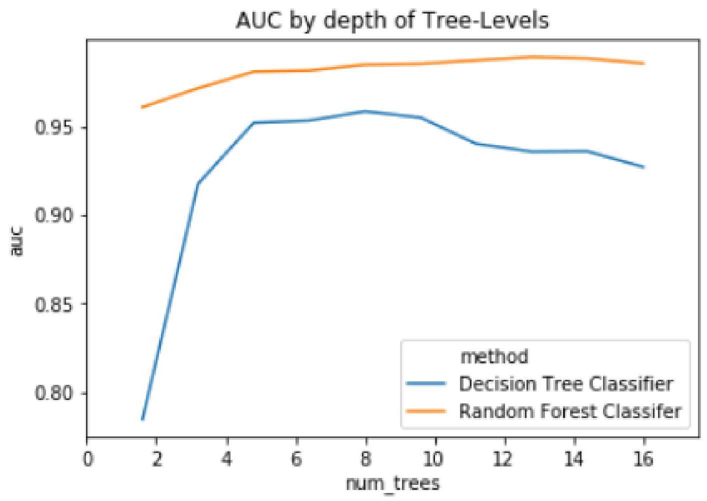
It is observed that the AUC of the Decision Tree Classifier with the Random Forest Classifier, it is clear that Random Forest performs better. However, we have provided "UNLIMITED Depth" to both classifiers which can easily result in extreme overfitting. Let's retrieve AUC score for both the method for varying number of "Depth" of Trees - ranging from 4 to 20.

```
Random_Forest = RandomForestClassifier(n_estimators=10, random_state=random_seed, max_features='sqrt')
Random_Forest = Random_Forest.fit(Xtrain, ytrain)

# Calculating AUC for Random Forest Classifer
y_test_pred = Random_Forest.predict(Xtest)
y_test_prob = Random_Forest.predict_proba(Xtest)[:,1]

#Calculating AUC for Classification Tree Model
clf_y_test_pred = classification.predict(Xtest)
clf_y_test_prob = classification.predict_proba(Xtest)[:,1]

print(f'Decision Tree : Test ROC AUC  Score: {roc_auc_score(ytest, clf_y_test_prob)}')
print(f'Random Forest : Test ROC AUC  Score: {roc_auc_score(ytest, y_test_prob)}')
```

```
Decision Tree : Test ROC AUC  Score: 0.9177935190036678
Random Forest : Test ROC AUC  Score: 0.9870168473660876
```

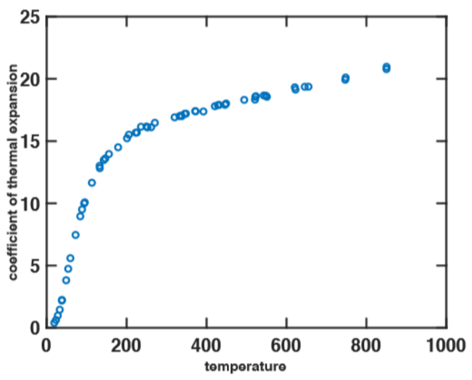
4) AUC is then plotted for both Decision Tree Classifier and Random Forest Classifier

```
def calc_roc_by_max_depth(Xtrain, Xtest, ytrain, ytest, max_depth, random_seed):
    # determine ROC for Decision Tree Classifer
    classification = tree.DecisionTreeClassifier(max_features='sqrt', max_depth=max_depth, random_state=ra
ndom_seed)
    classification = classification.fit(Xtrain, ytrain)
    classification_y_test_prob = classification.predict_proba(Xtest)[:,1]
    classification_auc = roc_auc_score(ytest, classification_y_test_prob)

    # determine ROC for Random Forest Classifier
    Random_Forest = RandomForestClassifier(n_estimators=10,max_depth=max_depth,max_features='sqrt',random_
state=random_seed)
    Random_Forest = Random_Forest.fit(Xtrain, ytrain)
    Random_Forest_y_test_prob = Random_Forest.predict_proba(Xtest)[:,1]
    Random_Forest_auc = roc_auc_score(ytest, Random_Forest_y_test_prob)

    return classification_auc, Random_Forest_auc

# Range for depth of trees from 2 to 20, calculate ROC using both methods
rec_list = []
for n in range(2,22,2):
    classification_auc,Random_Forest_auc = calc_roc_by_max_depth(Xtrain, Xtest, ytrain, ytest, n , random_
seed)
    #print("For depth = {} , clf_roc = {} rnf_roc = {} ".format(n,clf_roc,rnf_roc))
    rec_list.append([n,classification_auc, 'Decision Tree Classifier'])
    rec_list.append([n,Random_Forest_auc,'Random Forest Classifer'])

data_df = pd.DataFrame(rec_list, columns=['num_trees','auc','method'])
xlabels=[0,2,4,6,8,10,12,14,16,18,20]
ax = sns.lineplot(x = 'num_trees', y = 'auc', hue= 'method',data=data_df)
ax.set(xlim=(0, 22))
ax.set(title="AUC by depth of Tree-Levels")
ax.set_xticklabels(xlabels);
```



2. Nonlinear regression and cross-validation

The coefficient of thermal expansion y changes with temperature x. An experiment to relate y to x was done. Temperature was measured in degrees Kelvin. (The Kelvin temperature is the Celcius temperature plus 273.15). The raw data file is copper-new.txt.



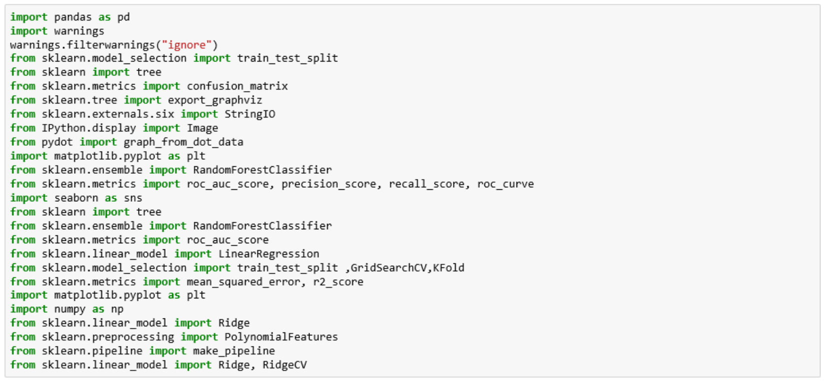
Question 2.1

1. (10 points) Perform linear regression on the data. Report the fitted model and the fitting error.

Solution 2.1

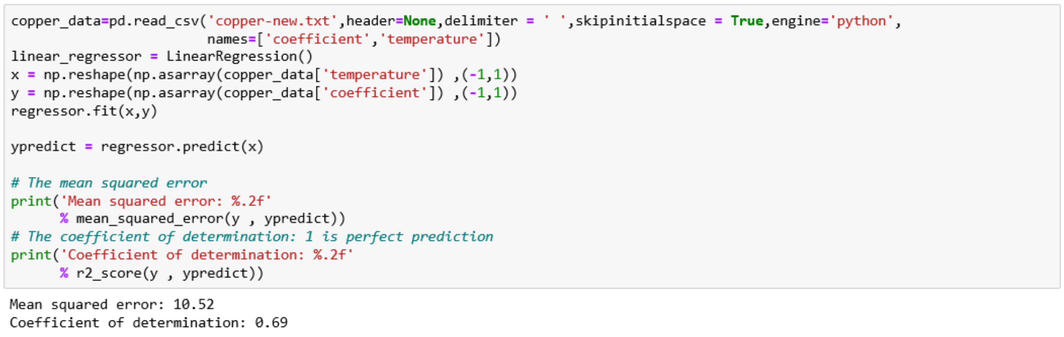
The steps given below were performed in order to solve these question

1) All the important modules pertaining to this HW was imported as shown in screenshot below



```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.metrics import confusion_matrix
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
from pydot import graph_from_dot_data
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, precision_score, recall_score, roc_curve
import seaborn as sns
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split ,GridSearchCV,KFold
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import Ridge
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import Ridge, RidgeCV
```

2) The copper-new.txt file is imported into a Pandas dataframe named copper_data. The first column is named <u>coefficient</u> and the second column is named <u>temperature</u>. Coeff is set as a response and temperature is set as predictor.
3) Linear regression is performed on this data as shown in the code below.



```
copper_data=pd.read_csv('copper-new.txt',header=None,delimiter = ' ',skipinitialspace = True,engine='python',
                        names=['coefficient','temperature'])
linear_regressor = LinearRegression()
x = np.reshape(np.asarray(copper_data['temperature']) ,(-1,1))
y = np.reshape(np.asarray(copper_data['coefficient']) ,(-1,1))
regressor.fit(x,y)

ypredict = regressor.predict(x)

# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y , ypredict))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y , ypredict))
```

```
Mean squared error: 10.52
Coefficient of determination: 0.69
```

4) Given below was the output

   <u>Mean squared error</u>: **10.52**

   <u>Coefficient of determination</u>: **0.69**

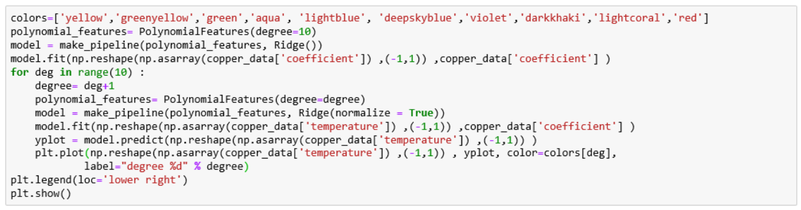
Question 2.2

2. (10 points) Perform nonlinear regression with polynomial regression function up to degree $n = 10$ and use ridge regression (see Lecture Slides for "Bias-Variance Tradeoff"). Write down your formulation and strategy for doing this, the form of the ridge regression.

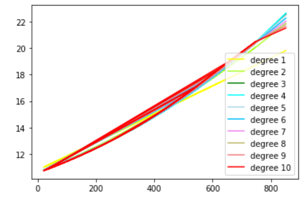
Solution 2.2

The steps given below were performed in order to solve these question

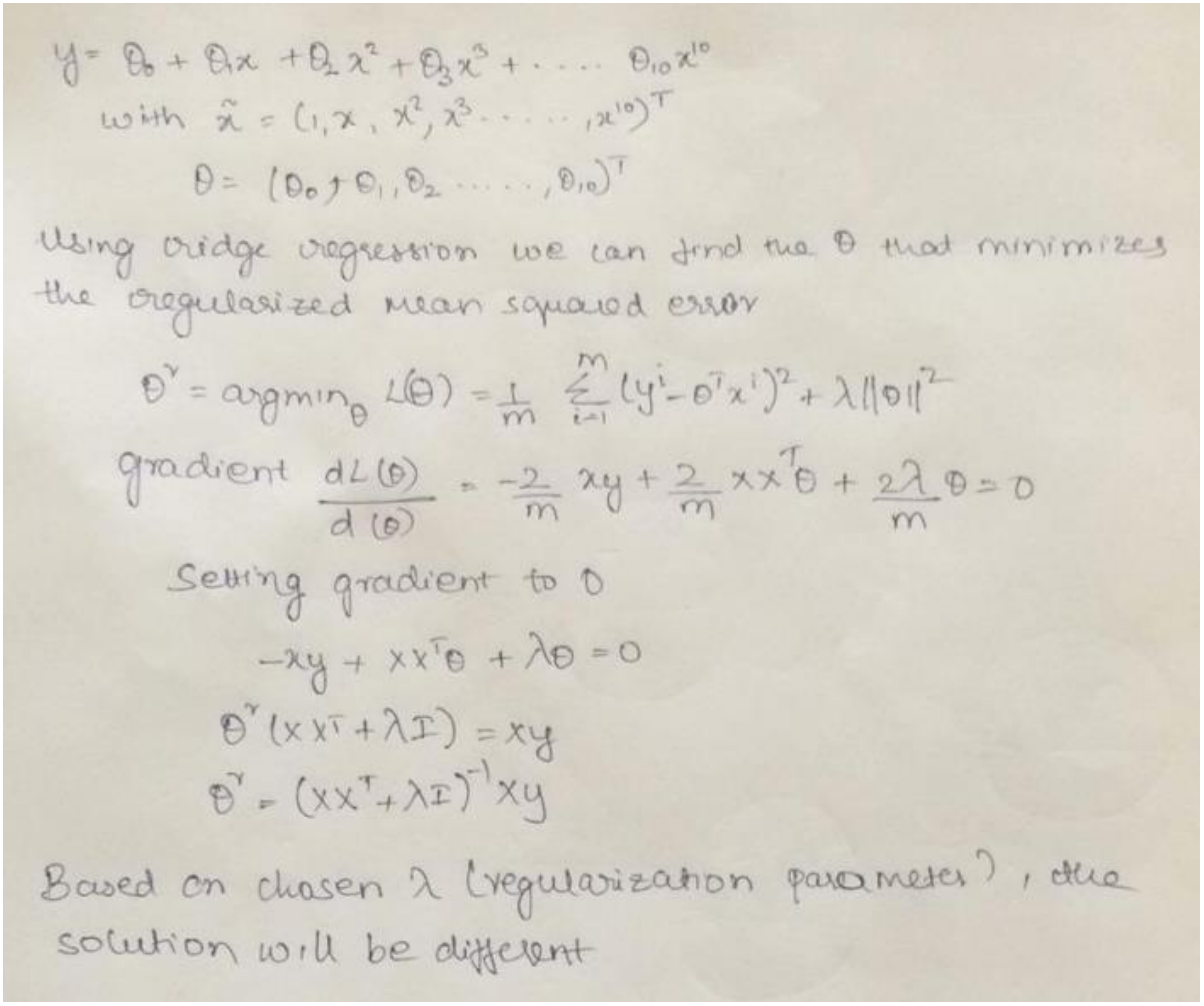
1) The copper-new.txt file is imported into a Pandas dataframe named copper_data. The first column is named coeff and the second column is named temperature. Coeff is set as a response and temperature is set as predictor.
2) Nonlinear regression with polynomial regression functions up to degree n = 10 was performed using Ridge Regression as shown in the code below



```
colors=['yellow','greenyellow','green','aqua', 'lightblue', 'deepskyblue','violet','darkkhaki','lightcoral','red']
polynomial_features= PolynomialFeatures(degree=10)
model = make_pipeline(polynomial_features, Ridge())
model.fit(np.reshape(np.asarray(copper_data['coefficient']) ,(-1,1)) ,copper_data['coefficient'] )
for deg in range(10) :
    degree= deg+1
    polynomial_features= PolynomialFeatures(degree=degree)
    model = make_pipeline(polynomial_features, Ridge(normalize = True))
    model.fit(np.reshape(np.asarray(copper_data['temperature']) ,(-1,1)) ,copper_data['coefficient'] )
    yplot = model.predict(np.reshape(np.asarray(copper_data['temperature']) ,(-1,1)) )
    plt.plot(np.reshape(np.asarray(copper_data['temperature']) ,(-1,1)) , yplot, color=colors[deg],
             label="degree %d" % degree)
plt.legend(loc='lower right')
plt.show()
```



3) Given below is the formulation strategy for the form of the Ridge Regression.



$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \dots \theta_{10} x^{10}$

with $\tilde{x} = (1, x, x^2, x^3 \dots\dots, x^{10})^T$

$\theta = (\theta_0, \theta_1, \theta_2 \dots\dots, \theta_{10})^T$

Using ridge regression we can find the θ that minimizes the regularized mean squared error

$$\theta^r = \operatorname{argmin}_\theta L(\theta) = \frac{1}{m}\sum_{i=1}^{m}(y^i - \theta^T x^i)^2 + \lambda\|\theta\|^2$$

gradient $$\frac{dL(\theta)}{d(\theta)} = -\frac{2}{m}xy + \frac{2}{m}xx^T\theta + \frac{2\lambda}{m}\theta = 0$$

Setting gradient to 0

$$-xy + xx^T\theta + \lambda\theta = 0$$

$$\theta^r(xx^T + \lambda I) = xy$$

$$\theta^r = (xx^T + \lambda I)^{-1}xy$$

Based on chosen λ (regularization parameter), the solution will be different

Question 2.3

3. (5 points) Use 5 fold cross validation to select the optimal regularization parameter λ. Plot the cross validation curve and report the optimal λ.

Solution 2.3

The steps given below were performed in order to solve these question

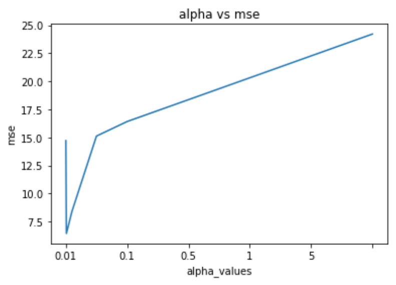
1) Coeff is set as a response and temperature is set as predictor in the copper_data dataset.
2) Lambda was tested with different values of alpha given below and optimal value of Lambda was found to be 0.01.

```
# Using 5-fold cross validation and degree of 10
X = np.reshape(np.asarray(copper_data['temperature']) ,(-1,1))
y = np.reshape(np.asarray(copper_data['coefficient']) ,(-1,1))
alpha_values=[ 0.001, 0.01, 0.1 , 0.5 ,1, 5 ]
kf = KFold(n_splits=5, shuffle=True, random_state=random_seed)
polynomial_features= PolynomialFeatures(degree=10)
ridge_mse=[]
for a in alpha_values:
    ridge_cv_model = Ridge(alpha=a, normalize=True)
    cv_mse=[]
    for train_idx,test_idx in kf.split(X):
        x_train_cv ,x_test_cv = X[train_idx], X[test_idx]
        y_train_cv ,y_test_cv = y[train_idx], y[test_idx]
        x_train_poly=polynomial_features.fit_transform(x_train_cv)
        x_test_poly=polynomial_features.fit_transform(x_test_cv)
        ridge_cv_model.fit(x_train_poly, y_train_cv)
        y_pred_cv=ridge_cv_model.predict(x_test_poly)
        theta=ridge_cv_model.coef_
        mse=np.sum((y_pred_cv - y_test_cv)**2)/len(y_pred_cv) + a*np.sum(theta**2)
        cv_mse.append(mse)

    ridge_mse.append((a, np.mean(cv_mse)))
df = pd.DataFrame(ridge_mse, columns =['alpha_values','mse'])
df2= df[df.mse == df.mse.min()]
Optimal_Lambda = df2.iloc[0]['alpha_values']
print("Optimal value of Lambda is:",Optimal_Lambda )
```

```
Optimal value of Lambda is: 0.01
```

3) The cross-validation curve was plotted with the optimal value of Lambda i.e. Lambda = 0.01 using the code below

```
xlabels=alpha_values
ax = sns.lineplot(x = 'alpha_values', y = 'mse',data=df)
ax.set(title="alpha vs mse")
ax.set_xticklabels(xlabels);
```



Observation: Looking at the plot above, the optimal value of Lambda is found to be 0.01.

Question 2.4

4. (5 points) Predict the coefficient at 400 degree Kelvin using both models. Comment on how would you compare the accuracy of predictions.

Solution 2.4

1) Coefficient for 400-degree temperature was predicted using the code below.

```
linear_regressor = regressor.predict(np.reshape([400],(-1,1)))
print('Linear Regression Coefficient: ',linear_regressor[0][0])
non_linear_regressor= model.predict(np.reshape([400],(-1,1)))
print('Non linear regressor with polynomial regression Coefficient: ',non_linear_regressor[0])
```

```
Linear Regression Coefficient:  15.897385088092761
Non linear regressor with polynomial regression Coefficient:  17.504654493738265
```

Given below are the coefficients calculated for both Linear and Non-Linear regressor with Polynomial Coefficient.

<u>Linear Regression Coefficient</u>: **15.897385088092761**

<u>Non-Linear regressor with polynomial regression Coefficient</u>: **14.866014785537029**

2) Accuracies were calculated for both the regression using the code below

```
linear_reg_pred = linear_regressor.predict(x)
print("Linear Regressor Accuracy: ",round(r2_score(y, linear_reg_pred,multioutput='variance_weighted')*100,2),'%')
non_linear_reg_pred = model.predict(x)
print("Non-Linear Regressor Accuracy:",round(r2_score(y, non_linear_reg_pred,multioutput='variance_weighted')*100,2),'%')
```

```
Linear Regressor Accuracy:  68.62 %
Non-Linear Regressor Accuracy: 53.16 %
```

Given below are the Accuracies calculated for both Linear and Non-Linear regressor with Polynomial Coefficient.
Linear Regressor Accuracy: **68.62 %**
Non-Linear Regressor Accuracy: **53.16 %**
In order to compare the accuracy of any prediction, we need to first calculate the R-Squared value of the regressor. The R-Squared value is the measure of Model's accuracy. For the above two regressors R-Squared value of both the regressors were calculated and it was observed that the linear regressor had higher accuracy as compared to Non-Linear regressor with Polynomial Regression.

3. Nonlinear regression and cross-validation

Consider a dataset with n data points (x_i, y_i), $x_i \in \mathbb{R}^p$, drawn from the following linear model:

$$y = x^T \beta^* + \epsilon,$$

where ϵ is a Gaussian noise and the star sign is used to differentiate the true parameter from the estimators that will be introduced later. Consider the regularized linear regression as follows:

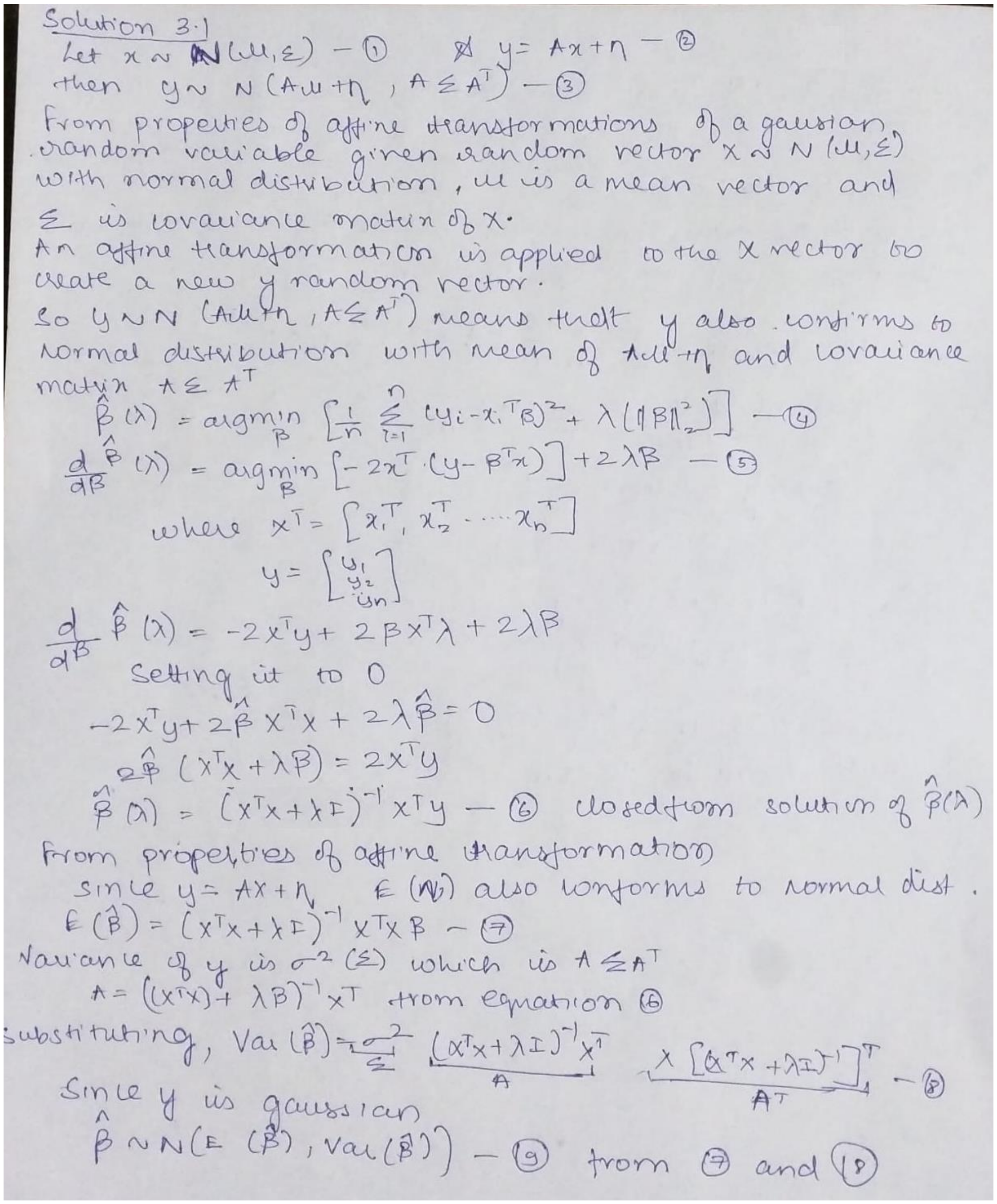
$$\hat{\beta}(\lambda) = \arg\min_{\beta} \left\{ \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \|\beta\|_2^2 \right\},$$

where $\lambda \geq 0$ is the regularized parameter. Let $X \in \mathbb{R}^{n \times p}$ denote the matrix obtained by stacking x_i^T in each row.

Question 3.1

1. **(10 points) Find the closed form solution for $\hat{\beta}(\lambda)$ and its distribution.**

Solution 3.1



Solution 3.1

Let $x \sim N(\mu, \Sigma)$ — ① ~~&~~ $y = Ax + \eta$ — ②

then $y \sim N(A\mu + \eta, A\Sigma A^T)$ — ③

From properties of affine transformations of a gausian random variable given random vector $x \sim N(\mu, \Sigma)$ with normal distribution, μ is a mean vector and Σ is covariance matrix of x.

An affine transformation is applied to the x vector to create a new y random vector.

So $y \sim N(A\mu + \eta, A\Sigma A^T)$ means that y also confirms to normal distribution with mean of $A\mu + \eta$ and covariance matrix $A\Sigma A^T$

$$\hat{\beta}(\lambda) = \underset{\beta}{\operatorname{argmin}} \left[\frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda (\|\beta\|_2^2)\right] \quad — ④$$

$$\frac{d}{d\beta} \hat{\beta}(\lambda) = \underset{\beta}{\operatorname{argmin}} \left[-2x^T \cdot (y - \beta^T x)\right] + 2\lambda\beta \quad — ⑤$$

where $x^T = [x_1^T, x_2^T \cdots x_n^T]$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

$$\frac{d}{d\beta} \hat{\beta}(\lambda) = -2x^T y + 2\beta x^T \lambda + 2\lambda\beta$$

Setting it to 0

$$-2x^T y + 2\hat{\beta} x^T x + 2\lambda\hat{\beta} = 0$$

$$2\hat{\beta}(x^T x + \lambda\beta) = 2x^T y$$

$$\hat{\beta}(\lambda) = (x^T x + \lambda I)^{-1} x^T y \quad — ⑥$$ closed form solution of $\hat{\beta}(\lambda)$

From properties of affine transformation

since $y = Ax + \eta$, $E(\hat{\beta})$ also conforms to normal dist.

$$E(\hat{\beta}) = (x^T x + \lambda I)^{-1} x^T x \beta \quad — ⑦$$

Variance of y is $\sigma^2(\Sigma)$ which is $A\Sigma A^T$

$A = ((x^T x) + \lambda\beta)^{-1} x^T$ from equation ⑥

substituting, $$\mathrm{Var}(\hat{\beta}) = \underbrace{\sigma^2}_{\Sigma} \underbrace{(x^T x + \lambda I)^{-1} x^T}_{A} \underbrace{x\,[(x^T x + \lambda I)^{-1}]^T}_{A^T} \quad — ⑧$$

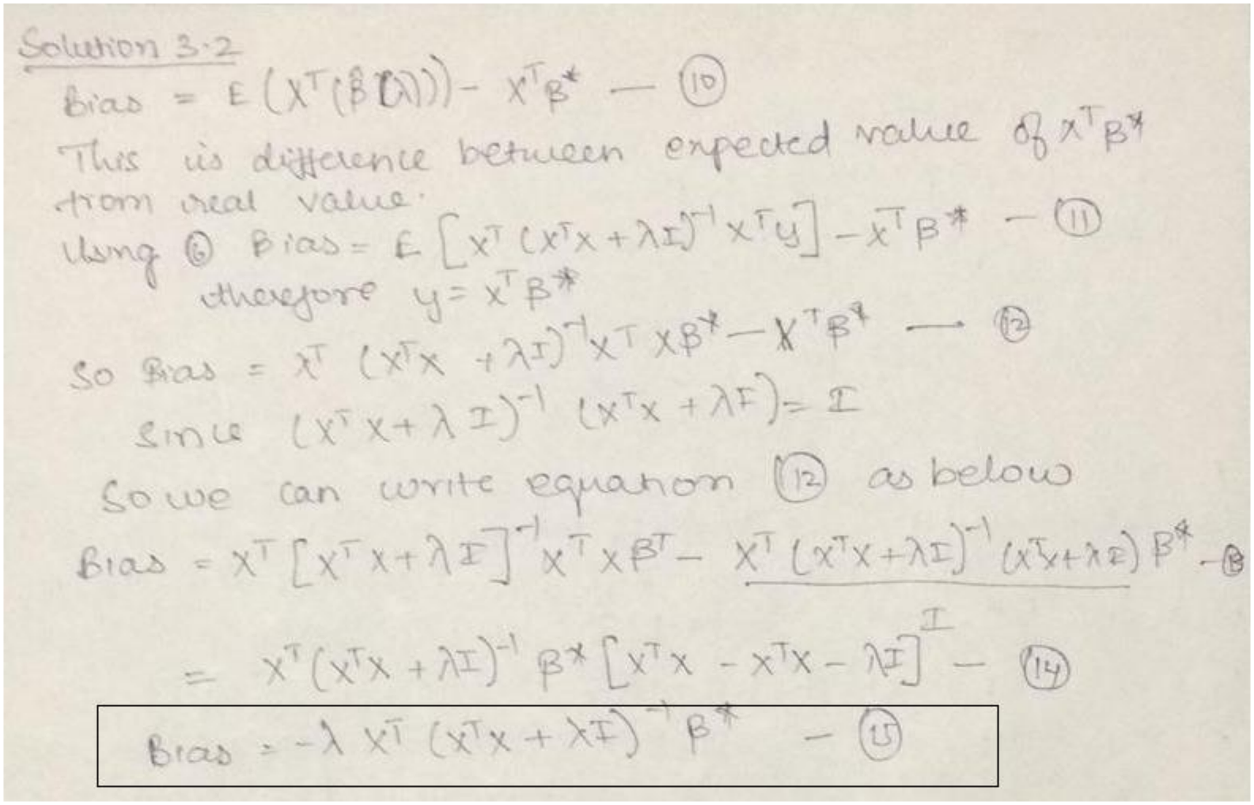
Since y is gaussian

$$\hat{\beta} \sim N(E(\hat{\beta}), \mathrm{Var}(\hat{\beta})) \quad — ⑨$$ from ⑦ and ⑧

Question 3.2

2. (10 points) Calculate the bias $\mathbb{E}[x^T\hat{\beta}(\lambda)] - x^T\beta^*$ as a function of λ and some fixed test point x.

Solution 3.2



Solution 3·2

$\text{Bias} = E(X^T(\hat{\beta}(\lambda))) - X^T\beta^* \quad — (10)$

This is difference between expected value of $x^T\beta^*$ from real value.

Using (6) $\text{Bias} = E\left[x^T(X^TX+\lambda I)^{-1}X^Ty\right] - x^T\beta^* \quad — (11)$

therefore $y = X^T\beta^*$

So $\text{Bias} = x^T(X^TX+\lambda I)^{-1}X^TX\beta^* - X^T\beta^* \quad — (12)$

since $(X^TX+\lambda I)^{-1}(X^TX+\lambda I) = I$

So we can write equation (12) as below

$\text{Bias} = X^T\left[X^TX+\lambda I\right]^{-1}X^TX\beta^T - \underbrace{X^T(X^TX+\lambda I)^{-1}(X^TX+\lambda I)}_{I}\beta^* \quad — (13)$

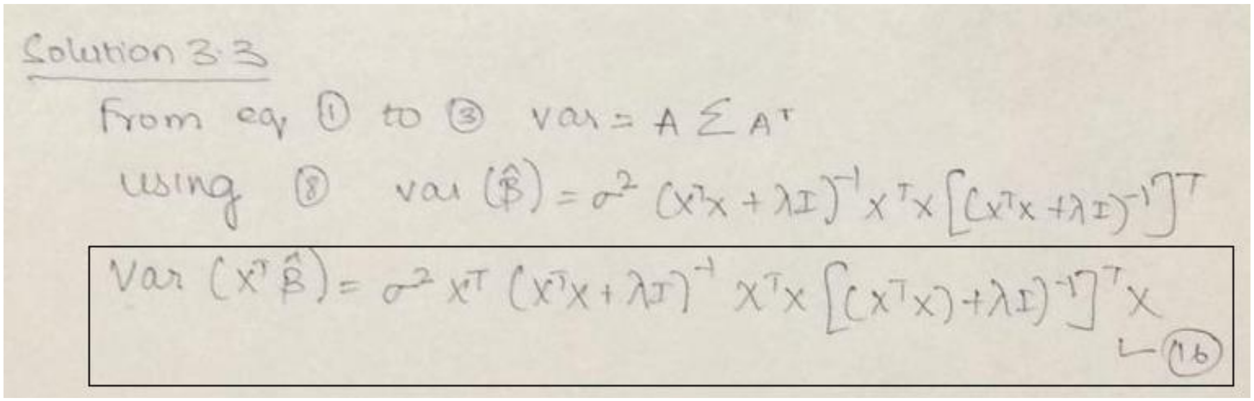
$= X^T(X^TX+\lambda I)^{-1}\beta^*\left[X^TX - X^TX - \lambda I\right] \quad — (14)$

$\boxed{\text{Bias} = -\lambda X^T(X^TX+\lambda I)^{-1}\beta^* \quad — (15)}$

Question 3.3

3. (10 points) Calculate the variance term $\mathbb{E}\left[\left(x^T\hat{\beta}(\lambda) - \mathbb{E}[x^T\hat{\beta}(\lambda)]\right)^2\right]$.

Solution 3.3



Solution 3·3

From eq (1) to (3) $\text{var} = A\Sigma A^T$

using (8) $\text{var}(\hat{\beta}) = \sigma^2(X^TX+\lambda I)^{-1}X^TX\left[(X^TX+\lambda I)^{-1}\right]^T$

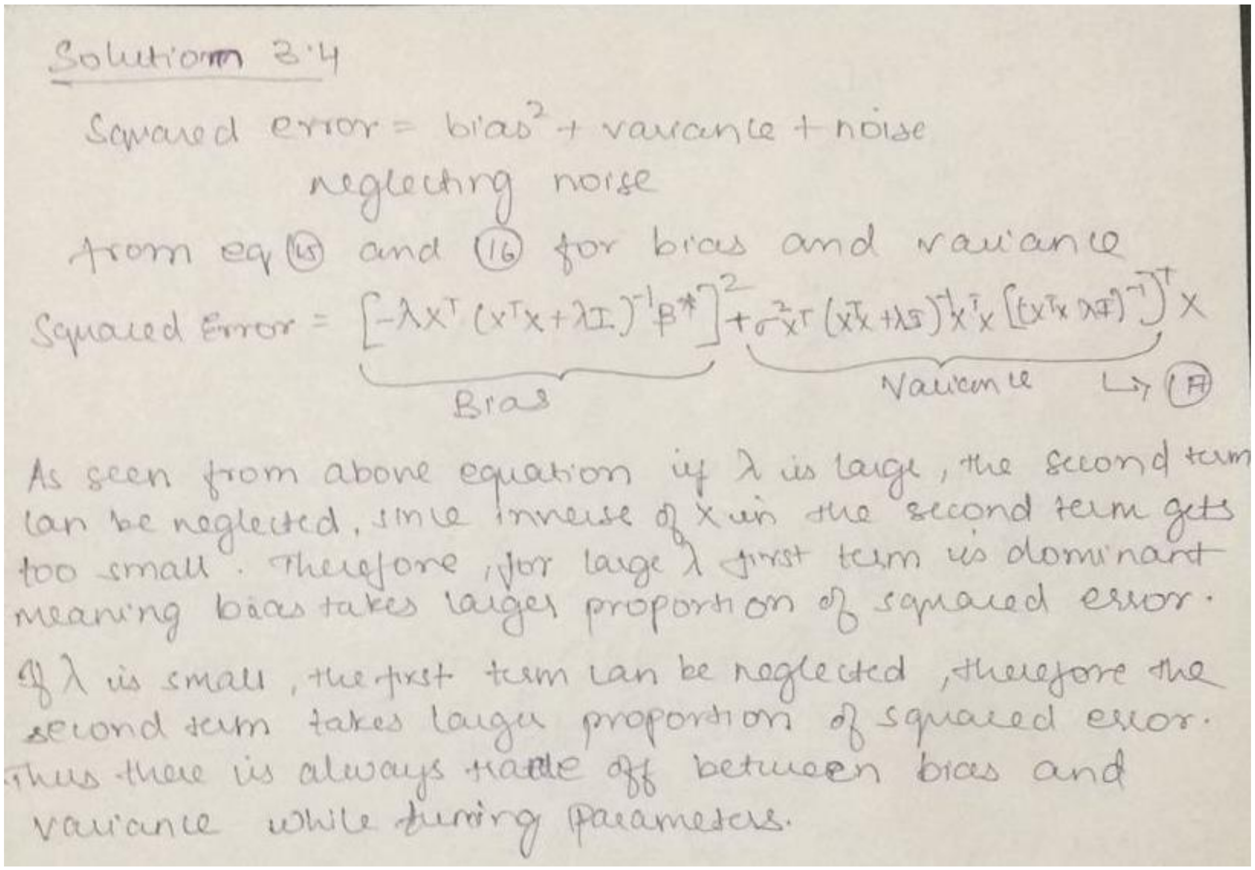
$\boxed{\text{Var}(X^T\hat{\beta}) = \sigma^2 X^T(X^TX+\lambda I)^{-1}X^TX\left[((X^TX)+\lambda I)^{-1}\right]^TX \quad — (16)}$

Question 3.4

4. (10 points) Use the results from parts (b) and (c) and the bias-variance decomposition to analyze the impact of λ in the squared error. Specifically, which term dominates when λ is small, and large, respectively?

(Hint.) Properties of an affine transformation of a Gaussian random variable will be useful throughout this problem.

Solution 3.4



Solution 3.4

Squared error = bias^2 + variance + noise

neglecting noise

from eq (15) and (16) for bias and variance

$$\text{Squared Error} = \underbrace{\left[-\lambda x^T (X^T X + \lambda I)^{-1} \beta^*\right]^2}_{\text{Bias}} + \underbrace{\sigma^2 x^T (X^T X + \lambda I)^{-1} X^T X \left[(X^T X + \lambda I)^{-1}\right]^T x}_{\text{Variance}} \hookrightarrow (17)$$

As seen from above equation if λ is large, the second term can be neglected, since inverse of X in the second term gets too small. Therefore, for large λ first term is dominant meaning bias takes larger proportion of squared error.

If λ is small, the first term can be neglected, therefore the second term takes larger proportion of squared error.

Thus there is always trade off between bias and variance while tuning parameters.