Xuhong Ye

4617395

1.

(a)

Because the indicator function I(•) takes value +1 if its argument is true, and -1 otherwise, c3(x)=I(x<+∞) always predicts +1.

And then, according to conditions of threshold classifier and indicator function, I listed the form of threshold of classifier below:

classifier	threshold		
	(-∞,a)	[a,b)	[b,+∞)
C1	-1	+1	+1
C2	+1	+1	-1
C3	+1	+1	+1
0.1c3	0.1	0.2	0.1
f	0.1	2.1	0.1

Therefore, we could know the set represented by f=c1+c2+0.1c3 is {0.1,2.1}.

(b)

According to the question, we could know that f>0 when x is in intervals (-2,-1)∪(1,2). Thus, I listed the form of threshold of classifier below:

classifier	threshold				
	(-∞,-2)	(-2,-1)	(-1,1)	(1,2)	(2,+∞)
C1=I(x>1)	+1	+1	-1	-1	-1
C2=I(x<2)					
C3=I(x<-1)	-1	+1	+1	+1	+1
C4=I(x>-2)					
f	0	2	0	0	0

Because the example model puts 0 in the case where the result is +1, I add C5=I(x<-∞) in the end, which is always equal to -1.

classifier	threshold				
	(-∞,-2)	(-2,-1)	(-1,1)	(1,2)	(2,+∞)
C1=I(x>1)	+1	+1	-1	-1	-1
C2=I(x<2)					
C3=I(x<-1)	-1	+1	+1	+1	+1
C4=I(x>-2)					

-C5=I(x<-∞)	+1	+1	+1	+1	+1
f	1	3	1	1	1

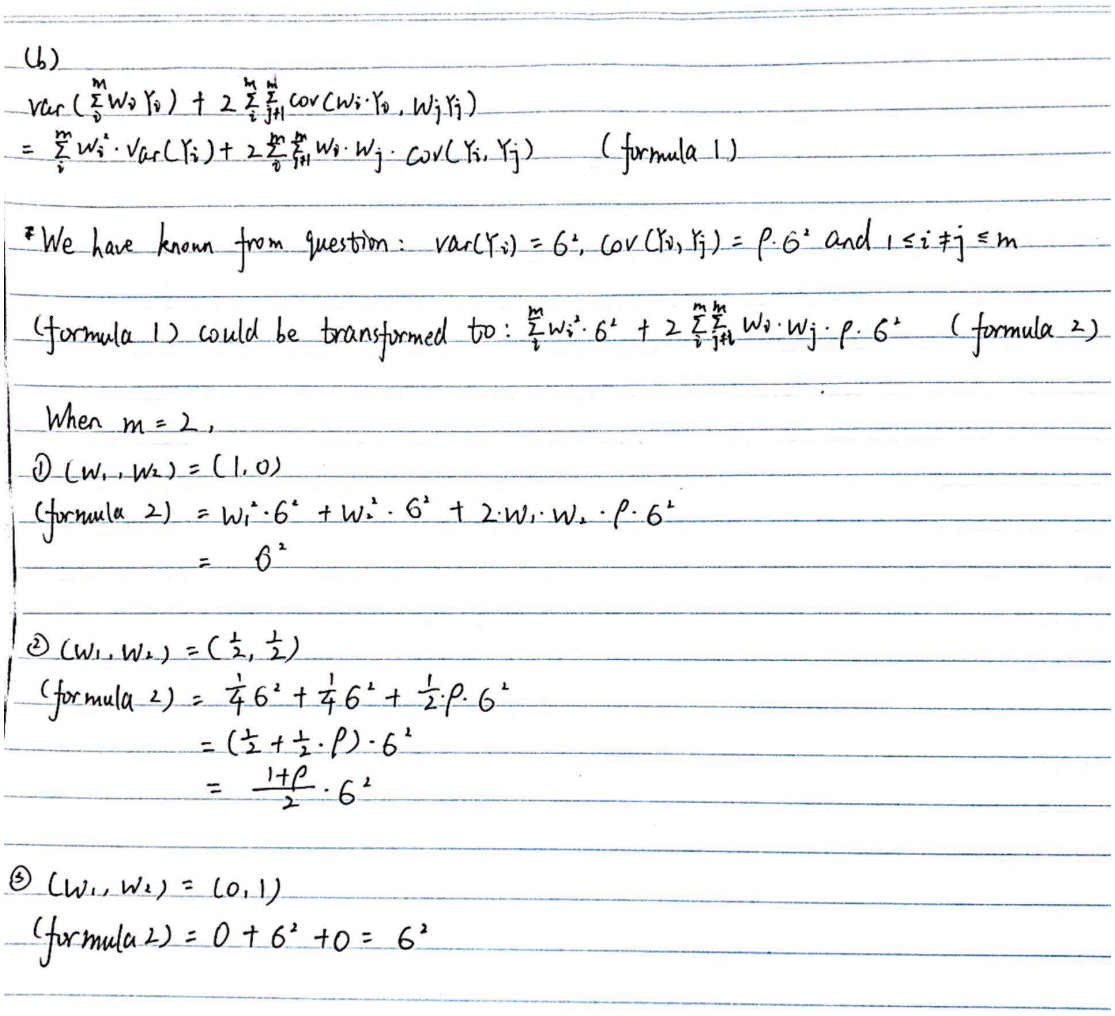
Therefore, a linear combination of threshold classifiers to represent two intervals $(-2,-1) \cup (1,2)$ is: f=c1+c2+c3+c4-c5, where c1=I(x>1),c2=I(x<2),c3=I(x<-1),c4=I(x>-2) and c5=I(x<-∞).

2.

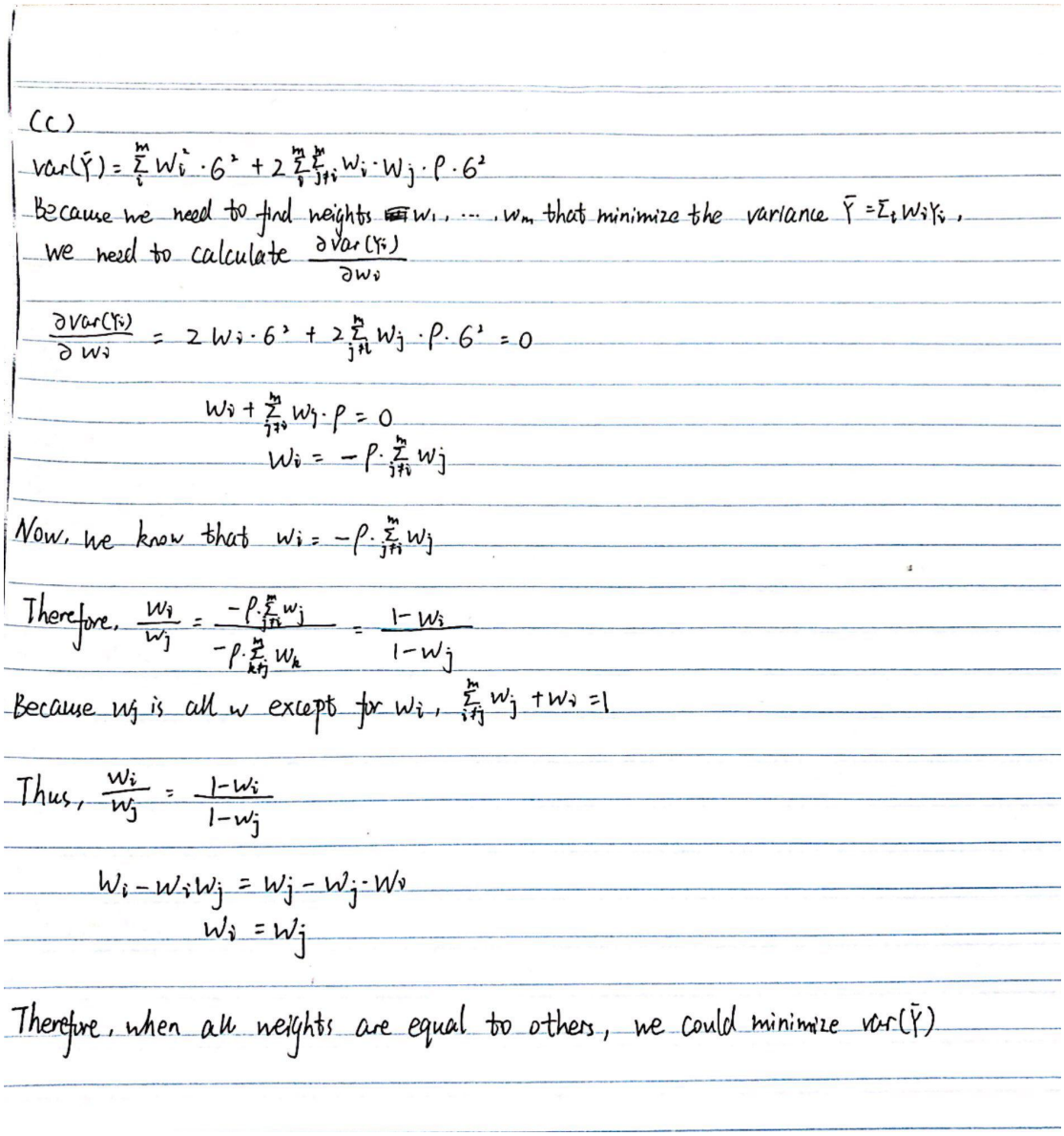
(a)

According to the question "assume that Y1, . . . , Ym are identically distributed with var(Yi) = σ^2 for all i", which means var(Yi) doesn't change as the model changes and is always constant. Therefore, we could know that Wagging has the same bias as each individual model.

(b)



(c)



(c)

$\text{var}(\bar{Y}) = \sum_i^m w_i^2 \cdot \sigma^2 + 2\sum_i^m \sum_{j \neq i} w_i \cdot w_j \cdot \rho \cdot \sigma^2$

Because we need to find weights $w_1, \dots, w_m$ that minimize the variance $\bar{Y} = \Sigma_i w_i Y_i$,

we need to calculate $\frac{\partial \text{var}(Y_i)}{\partial w_i}$

$\frac{\partial \text{var}(Y_i)}{\partial w_i} = 2 w_i \cdot \sigma^2 + 2\sum_{j \neq i} w_j \cdot \rho \cdot \sigma^2 = 0$

$w_i + \sum_{j \neq i} w_j \cdot \rho = 0$

$w_i = -\rho \cdot \sum_{j \neq i} w_j$

Now, we know that $w_i = -\rho \cdot \sum_{j \neq i} w_j$

Therefore, $\frac{w_i}{w_j} = \frac{-\rho \cdot \sum_{j \neq i} w_j}{-\rho \cdot \sum_{k \neq j} w_k} = \frac{1 - w_i}{1 - w_j}$

Because w_j is all w except for w_i, $\sum_{i \neq j}^m w_j + w_i = 1$

Thus, $\frac{w_i}{w_j} = \frac{1 - w_i}{1 - w_j}$

$w_i - w_i w_j = w_j - w_j \cdot w_i$

$w_i = w_j$

Therefore, when all weights are equal to others, we could minimize $\text{var}(\bar{Y})$

3.

(a)

(1)

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

housing = fetch_california_housing()
x=pd.DataFrame(housing.data,columns=housing.feature_names)
y=housing.target
print(x.shape)
```

The output is (20640, 8), so we could know that the value of d is 8.

(2)

```
random_seed=42
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,random_state=random_seed)
print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

I set the number of random seed is 42. And then, the outputs of x_train, x_test, y_train, y_test are (14448, 8), (6192, 8), (14448,) and (6192,) separately.

(b)

(1)From question(a), we could know that the shapes of x_train and x_test are (14448, 8) and (6192, 8) after using train_test_split() function. Therefore, the value of m is 8, which is equal to the value of d. The reason why this result happened is that one of parameters of train_test_split()function is max_features and the default value of it is "auto", which means that max_features=n_features. Thus, the value of m is equal to d.

(2)

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

rfr = RandomForestRegressor(n_estimators=100,random_state=random_seed)
rfr.fit(x_train, y_train)
y_pred_train,y_pred_test=rfr.predict(x_train),rfr.predict(x_test)
train_mse=mean_squared_error(y_train,y_pred_train) #0.03698995328644075
test_mse=mean_squared_error(y_test,y_pred_test) #0.2569942567398473
print(train_mse,test_mse)
```

I use mean_squared_error() function to measure accuracy. The mean squared error of an estimator measures the average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value. a smaller MSE means otherwise and it is definitely the preferred and/or desired choice as it shows that your data values are dispersed closely to its central moment (mean); which is usually great. The outputs of the training and test mse are 0.03698995328644075 and 0.2569942567398473 respectively. This means that the average squared difference of trian and test between the estimated values and the actual values are relatively small.

(c)

(1)

```
def get_correlation(rfr,test):
    base_learners=rfr.estimators_ #list of decission tree regressor
```

```
    base_predictions=[]
    for i in range(len(base_learners)):
        base_predictions.append(base_learners[i].predict(test))
    base_predictions=np.array(base_predictions)
    corr=np.corrcoef(base_predictions)

    return corr

corr=get_correlation(rfr,x_test)
print(corr)
```

The output of all the pairwise correlations between the test set predictions of the 100 trees is:

```
[[1.         0.75771861 0.76903759 ... 0.74834924 0.75885034 0.77774916]
 [0.75771861 1.         0.75988204 ... 0.74944788 0.75795134 0.76498828]
 [0.76903759 0.75988204 1.         ... 0.75258947 0.7667629  0.77247691]
 ...
 [0.74834924 0.74944788 0.75258947 ... 1.         0.74320362 0.76258888]
 [0.75885034 0.75795134 0.7667629  ... 0.74320362 1.         0.77001229]
 [0.77774916 0.76498828 0.77247691 ... 0.76258888 0.77001229 1.        ]]
```

The details of output could be seen in the “correlation.csv”.

(2)

```
avg_correlation=[]
avg_correlation.append(np.mean(corr))
print(avg_correlation)
```

The average of all the pairwise correlations between the test set predictions of the 100 trees is:

```
[0.7652796942329071]
```

(d)

```
train_MSE=[]
test_MSE=[]
avg_corr=[]

for m in range(1,9):

    rfr1=RandomForestRegressor(n_estimators=100,random_state=random_seed,max_features=m)
    rfr1.fit(x_train,y_train)
    y_train_pred=rfr1.predict(x_train)
    train_MSE.append(mean_squared_error(y_train_pred,y_train))
```

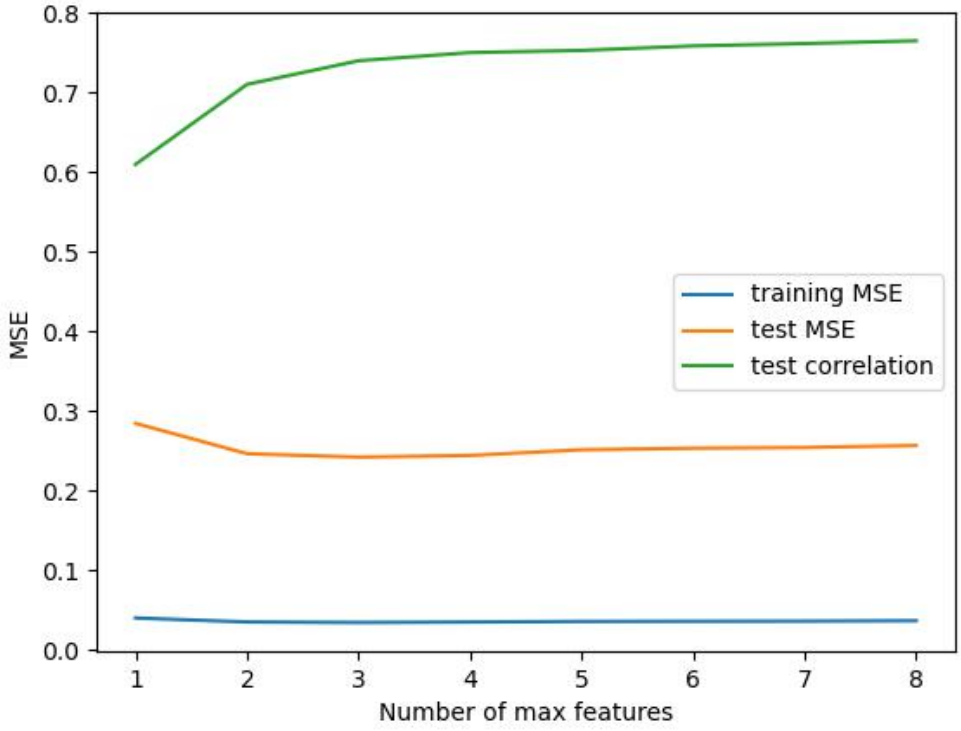
```
        y_test_pred=rfr1.predict(x_test)
        test_MSE.append(mean_squared_error(y_test_pred,y_test))

        corr=get_correlation(rfr1,x_test)
        avg_corr.append(np.mean(corr))

result={"training accuracy":train_MSE,"testing accuracy":test_MSE,"average correlation":avg_corr}
result_df=pd.DataFrame(result)
result_df.to_csv("result.csv")
```

I tabulated the training and test accuracies, and the average correlations for all m values, which is listed below:

	training accuracy	testing accuracy	average correlation
0	0.040372228	0.284834419	0.610075047
1	0.035390846	0.24668487	0.710542084
2	0.034710454	0.242433586	0.740238506
3	0.035305724	0.244514728	0.750476323
4	0.036032001	0.251657464	0.753193766
5	0.036273677	0.253629474	0.758850978
6	0.036443317	0.254527552	0.761802971
7	0.036989953	0.256994257	0.765279694

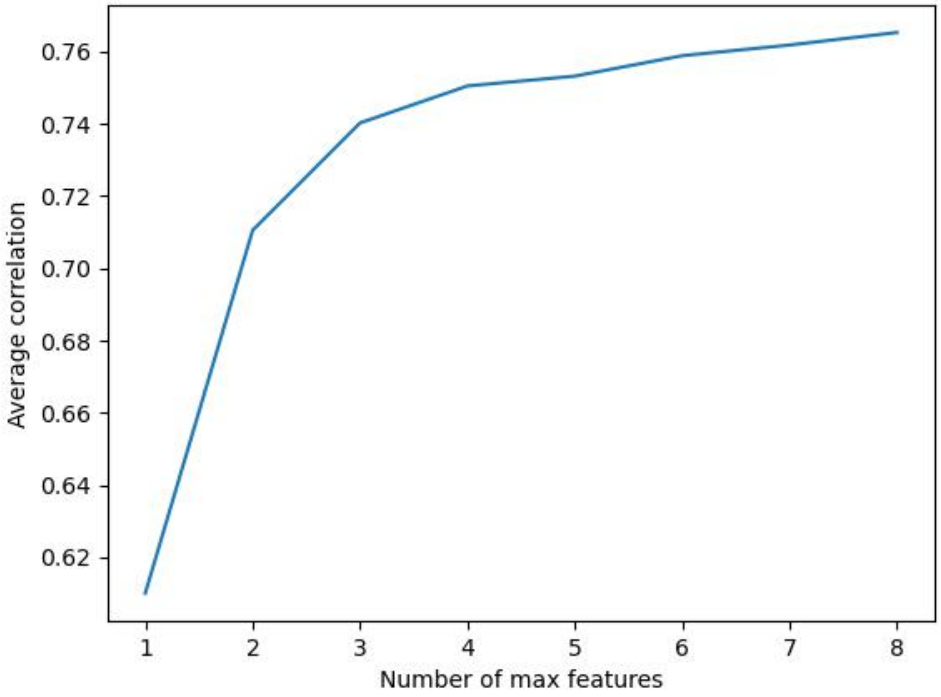
```
plt.plot(range(1,9),train_MSE)
plt.plot(range(1,9),test_MSE)
plt.plot(range(1,9),avg_corr)
plt.legend(["training MSE","test MSE","test correlation"])
plt.xlabel("Number of max features")
plt.ylabel("MSE")
plt.show()
```

I plot the training and test accuracies against m in a single figure, which is listed below:

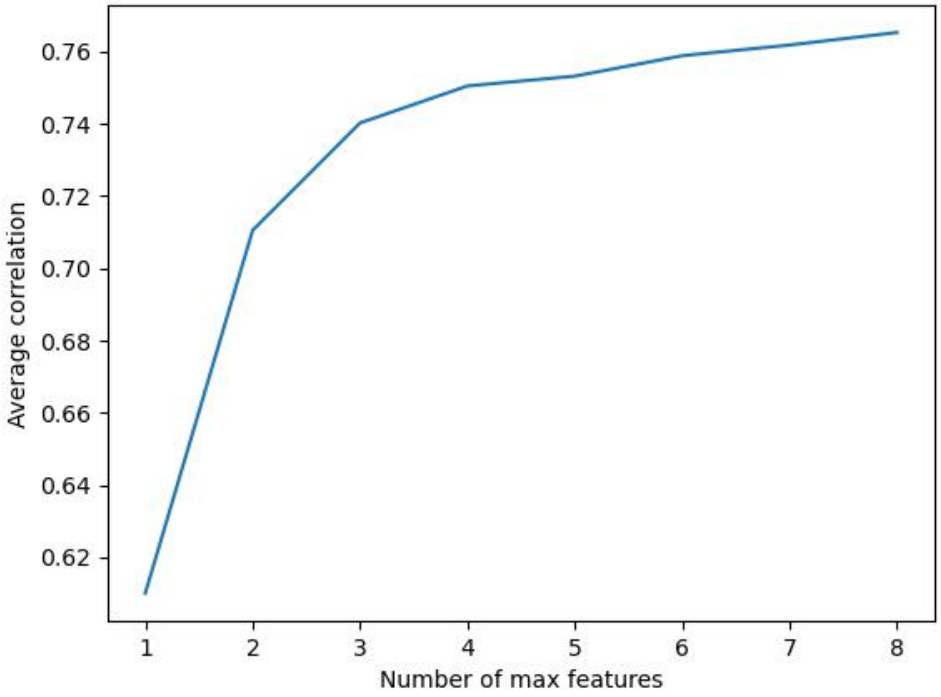



```
plt.plot(range(1,9),avg_corr)
plt.xlabel("Number of max features")
plt.ylabel("Average correlation")
plt.show()
```

I plot the average correlation against m in another figure, which is listed below:



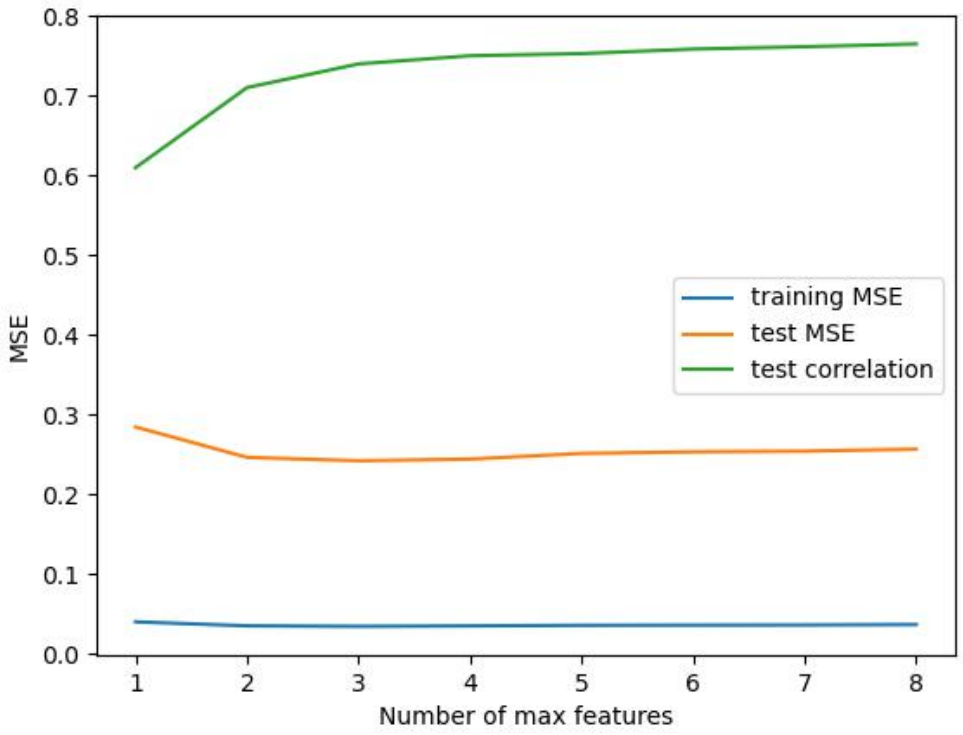
(e)



From the plot of the average correlation against m I listed above, we could find that

the average correlation increases with the increase of m. If we use m=1 to build 8 decision trees, these 8 decision trees will most likely use different features.Thus, these 8 trained decision tree are completely different, which means that the correlation between them is very low. If we use m=8 to build 8 decision trees, these 8 decision trees are very similar to others, because some training datasets overlap when training them.

(f)
As far as I am concerned, I think the claim of this data scientist is false.



The formula of expected prediction error (EPE):

$$\overbrace{\mathbb{E}\left((\tilde{Y}-Y)^2\right)}^{\text{expected prediction error}} = \overbrace{\mathbb{E}\left((\tilde{Y}-\mathbb{E}(\tilde{Y}))^2\right)}^{\text{variance}} + \overbrace{\left(\mathbb{E}(\tilde{Y})-\mathbb{E}(Y)\right)^2}^{\text{bias (squared)}} + \overbrace{\mathbb{E}\left((Y-\mathbb{E}(Y))^2\right)}^{\text{irreducible noise}}$$

According to the plot of the training and test accuracies against m I listed above, we could find that when m=1, the correlation and variance between machine learners is the lowest, but the bias is the highest. However, according to the formula of EPE, when we select a appropriate model, we have to think about not only the relatively low variance but also the relatively low bias.

4.
(a)
from sklearn.datasets import load_digits

```
from sklearn.model_selection import train_test_split

digits = load_digits()
random_seed = 999
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.3, random_state=random_seed)
```

(b)

First, we should find out the index of OOB data from sample. And then, we could find the corresponding OOB data from the dataset according to these indexes. Thirdly, we use these OOB data to calculate probability of predicted result. Then, we construct a matrix , where each row is the predicted value of each data and each column is the predicted result of the current data in each model. This matrix contains a fixed amount of data. After that, after training with each machine learner, we will find out the probability of predicted result from the corresponding OOB data, and then we add these probabilities into the matrix according to the index of OOB data. Therefore, the amount of memory used should be at most within a constant factor of that for the original dataset size, although we introduce additional variables or data structures if needed. And then, the non-zero data in the matrix represents the current OOB value. Calculate accuracy based on current OOB data and predicted OOB data. After that, we use 1 minus accuracy to get OOB error. Finally, when the number of model is at least 10, we could stop training when the mean of oob error of the current model is greater than that of its previous five models.

(c)

I implement the fit function according to the method and thought which I have already described in question(b). I listed code below:

```
from sklearn.metrics import accuracy_score

    def fit(self, X, y, random_state=None):
        if random_state:
            np.random.seed(random_state)

        self.best_n = 0

        probs_oob = None
        oob_pred_total = np.zeros((X.shape[0], 10))

        for i in range(self.n_estimators):
            estimator = clone(self.base_estimator_)

            # construct a bootstrap sample
            bst_index = np.random.choice(range(X.shape[0]), X.shape[0],
```

```
replace=True)
                oob_index = np.setdiff1d(range(X.shape[0]), bst_index)
                bst_X = X[bst_index, :] #taining data features
                bst_y = y[bst_index] #training data target

                # train on bootstrap sample
                estimator.fit(bst_X, bst_y) #trained bootstrap sample
                self.estimators_.append(estimator) #list of decission tree regressor

                # compute OOB error
                oob_X = X[oob_index, :]
                # oob_y = y[oob_index]
                oob_pred = estimator.predict_proba(oob_X) #predict result of y

                for j, index in enumerate(oob_index):
                    oob_pred_total[index] += oob_pred[j]

                current_oob = list(set(np.nonzero(oob_pred_total)[0]))
                # print(len(current_oob))
                current_oob_pred = np.argmax(oob_pred_total[current_oob,:],
axis=1)
                oob_error = 1 - accuracy_score(y[current_oob], current_oob_pred) #
replace ... with your code
                self.oob_errors_.append(oob_error)

                # save the OOB error and the new model
                self.oob_errors_.append(oob_error)
                self.estimators_.append(estimator)

                # stop early if smoothed OOB error increases (for the purpose of
                # this problem, we don't stop training when the criterion is
                # fulfilled, but simply set self.best_n to (i+1)).
                if (self.best_n == 0) and (i >= 10 and
np.mean(self.oob_errors_[i:i-5:-1]) >= np.mean(self.oob_errors_[i-5:i-10:-1])): #
replace OOB criterion with your code
                    self.best_n = (i+1)
```

(d)

```
    def errors(self, X, y):
        '''
        Parameters
        ----------
        X: an input array of shape (n_sample, n_features)
        y: an array of shape (n_sample,) containing the classes for the input
```

examples

```
        Returns
        ------
        error_rates: an array of shape (n_estimators,), with the error_rates[i]
        being the error rate of the ensemble consisting of the first (i+1)
        models.
        '''
        error_rates = []
        # compute all the required error rates
        estimator_predictions = []

        for i, estimator in enumerate(self.estimators_):
            prob_pred = estimator.predict_proba(X)
            if i == 0:
                estimator_predictions.append(prob_pred)
            else:
                estimator_predictions.append(prob_pred +
estimator_predictions[i - 1])

        for i in range(len(self.estimators_)):
            class_pred = np.argmax(estimator_predictions[i], axis=1)
            accuracy = accuracy_score(y, class_pred)
            error = 1 - accuracy
            error_rates.append(error)
        return error_rates
```

(e)

(1)

```
base_estimator = DecisionTreeClassifier(random_state=random_seed)
bagging_estimator = OOBaggingClassifier(base_estimator, 100)
bagging_estimator.fit(X_train, y_train, random_seed)
print(bagging_estimator.best_n)
```

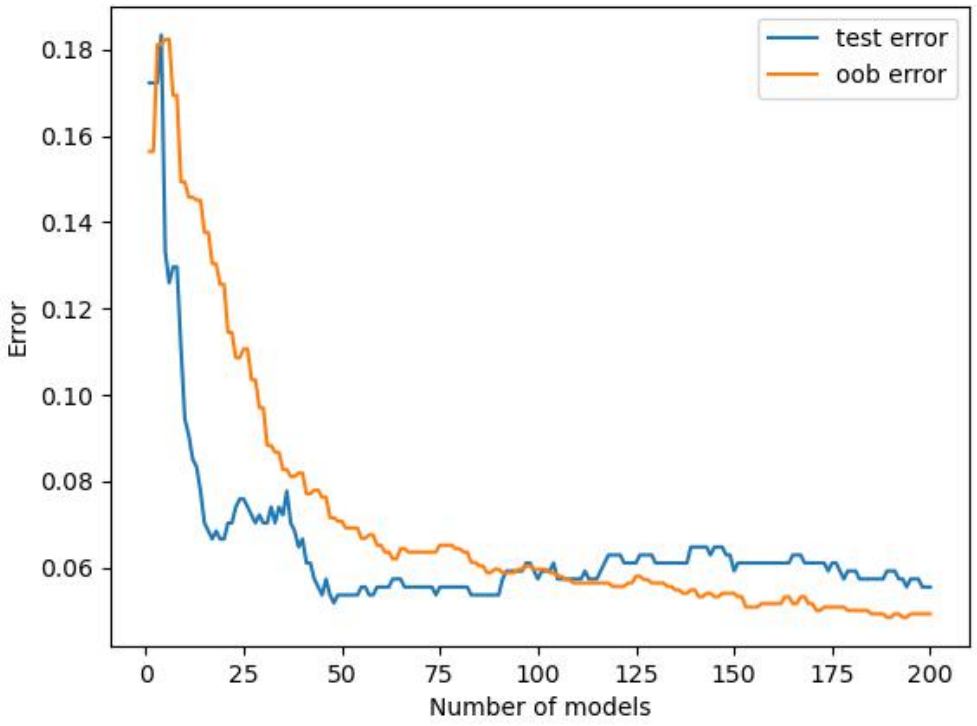
The output of the number of basis models chosen by the OOB error method is 69.

(2)

```
test_error=bagging_estimator.errors(X_test,y_test)
oob_error=bagging_estimator.oob_errors_

plt.plot(range(1,201),test_error)
plt.plot(range(1,201),oob_error)
plt.legend(["test error","oob error"])
plt.xlabel("Number of models")
```

```
plt.ylabel("Error")
plt.show()
```

The plot of the OOB error and the test error against the number of models (from 1 to 200) is:

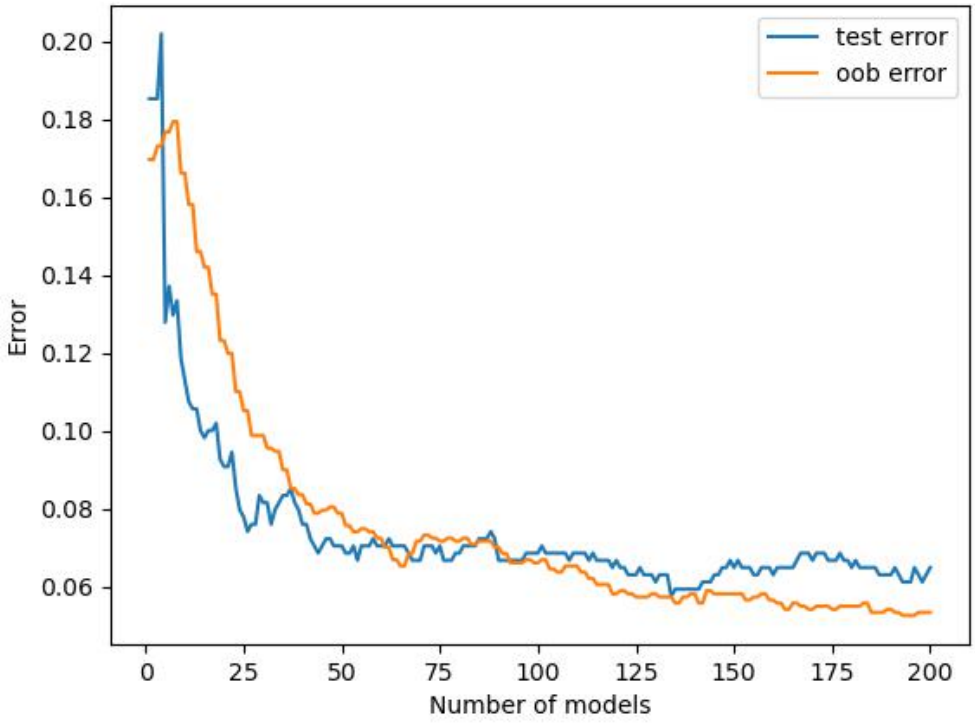


From the plot I listed above, we could know that the trends of test error and oob error are very similar because their trends are both down before the number of model is 100 and oob error always greater than test error. However, after m=100, the trend of test error is leveling off, but the trend of oob error is still down and oob error always lower than test error.

(f)

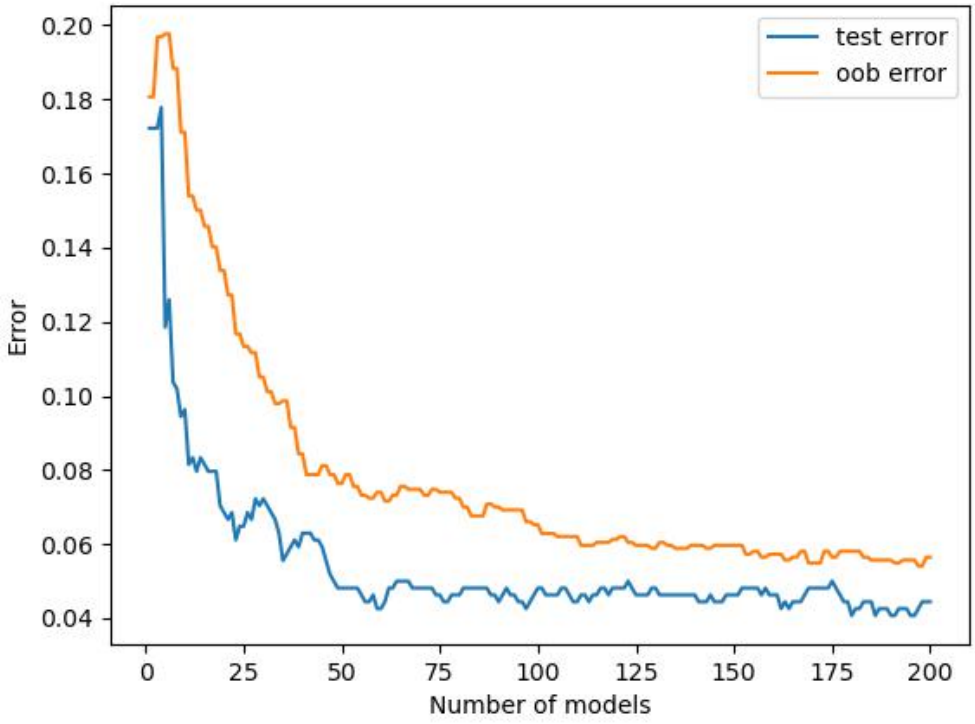
(1) When I set random seeds equal to 111, the number of basis models chosen by the OOB error method is 70.

The plot of the OOB error and the test error against the number of models (from 1 to 200) is:



(2) When I set random seeds equal to 555, the number of basis models chosen by the OOB error method is 66.

The plot of the OOB error and the test error against the number of models (from 1 to 200) is:



Compared three different outputs when random seeds equal to 111, 555 and 999 respectively, we could find that the range of best_n is roughly from 50 to 70.

The trends of test error and oob error when random seeds equal to 111 and 999 are very similar, they are both down before the number of model is 100 and oob error always greater than test error. However, after m=100, the trend of test error is leveling off, but the trend of oob error is still down and oob error always lower than test error.

But when random seeds equal to 555, the trends of test error and oob error are both down and oob error always greater than test error.