BDA 502 -ASSIGNMENT 2

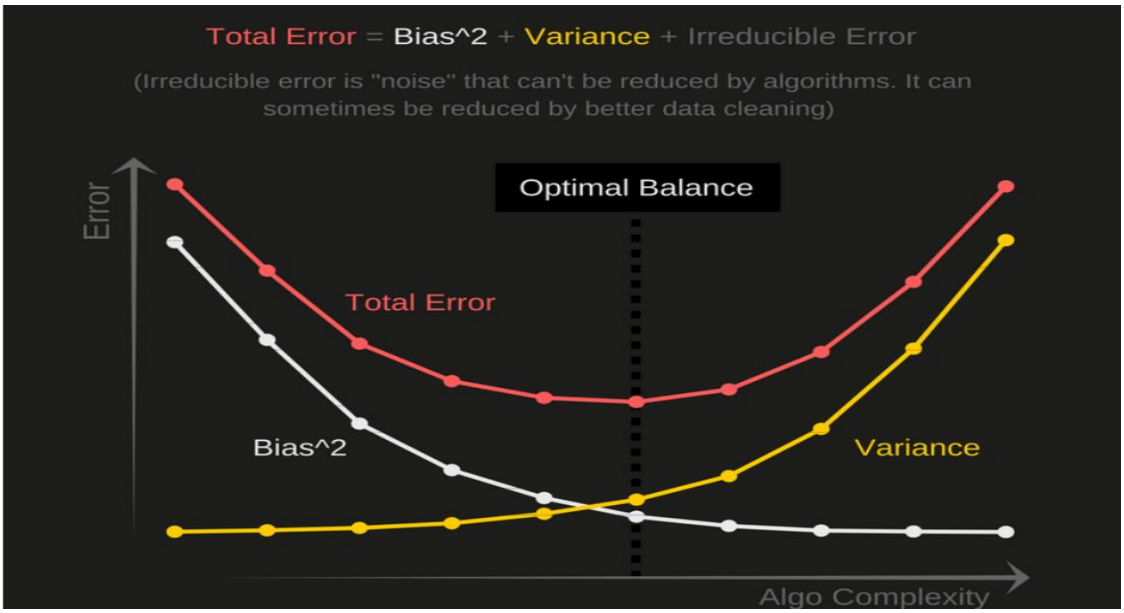
1. **Please discuss the tension between bias and variance during developing a machine learning model. What is the difference between them and what is the trade-off? How would you classify decision trees and linear regression regarding their level of bias.**

In supervised machine learning methods, algorithm learns a model from training data. There are 3 types of error on these methods: bias error, variance error and irreducible error.

A simple definition, **bias** is the difference between predictions of model and the true values. Bias is the model's assumption to make the target function easier to learn. So, high bias suggests more assumptions about the form of the target function while low bias suggest less assumptions. Example of high bias ML algo's: Linear Regression and Logistic Regression and example of low bias ML algo's: KNN, SVM, Decision trees. An example of bias error; if we try to apply linear regression on non-linear dataset, the model does not fit on it; Result will be **under-fitting**. Or, if we have highly correlated two features on linear regression model, they will cause over-fitting because of their high weight of bias. **(high bias causes under-fitting!)**

When we look at **variance**, it is the amount of the change on estimation of target function if train data is changed. Ideally, variance should not keenly change when train data is changed. (I mean from same dataset, different train data.) Low variance suggests small changes to estimate of the target with changes to the training data while high variance suggest large changes. So, **high variance causes overfitting!** Example of low-variance ML algo's: Lienar and Logistic Regression and example of high variance ML algo's: KNN,SVM Decision trees.

The goal of any supervised machine learning algorithm is low bias with low variance for good prediction. While increasing the bias causes decreasing the variance (or vice versa) there is trade-off when they balance due to some methods which are varied according to ML algorithm.



Source for image: https://elitedatascience.com/bias-variance-tradeoff

2. **You are expected to provide the descriptive information about the iris dataset. How many classes, cases and potential features are there? Please find the average values for each class regarding different features. Thus, we will have a rough idea about the potential differences about the classes. A manually made table will also be sufficient but you can also code it.**

#how many potential features? Due to following code chunk and output; 4 potential features; sepal_length, sepal_width, petal_length, petal_width.

```
iris = pd.read_csv("iris.csv")
print(iris.info())
```

```
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
None
```

```
print(iris.describe())
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
print(iris.species.unique())
['setosa' 'versicolor' 'virginica']
```

#record count per each specie:

```
Species_cnt = iris.groupby('species',as_index= False).count()
print(species_cnt)
```

	species	sepal_length	sepal_width	petal_length	petal_width
0	setosa	50	50	50	50
1	versicolor	50	50	50	50
2	virginica	50	50	50	50

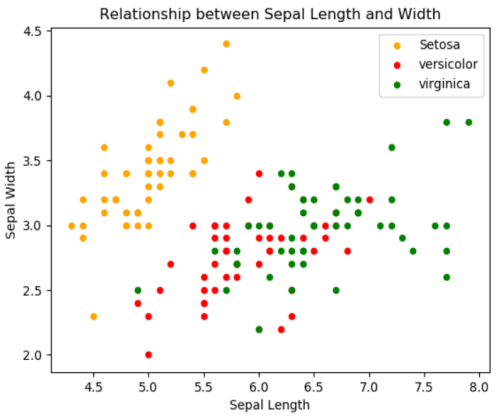
#record count per each specie:

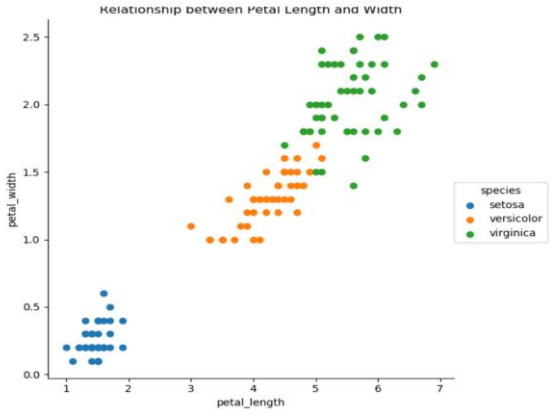
```
species = iris.groupby('species',as_index= False).mean()
print(species)
```

```
...
      species  sepal_length  sepal_width  petal_length  petal_width
0      setosa         5.006        3.418         1.464        0.244
1  versicolor         5.936        2.770         4.260        1.326
2   virginica         6.588        2.974         5.552        2.026
```

#draw the relationship between Sepal length and width due to species:

```
iris_df=iris
#Relationship between the Sepal Length and Width using scatter plot
ax =
iris_df[iris_df.species=='setosa'].plot.scatter(x='sepal_length',y='sepal_width'
,color='orange', label='Setosa')
iris_df[iris_df.species=='versicolor'].plot.scatter(x='sepal_length',y='sepal_wi
dth',color='white', label='versicolor',ax=ax)
iris_df[iris_df.species=='virginica'].plot.scatter(x='sepal_length',y='sepal_wid
th',color='green', label='virginica', ax=ax)
ax.set_xlabel("Sepal Length")
ax.set_ylabel("Sepal Width")
ax.set_title("Relationship between Sepal Length and Width")
plt.show()

#Relationship between the Petal Length and Width using seaborn
sns.FacetGrid(iris_df, hue="species", size=6) \
   .map(plt.scatter, "petal_length", "petal_width") \
   .add_legend()
plt.title("Relationship between Petal Length and Width")
```





3. **Please run the code for iris dataset for linear regression. You should select one feature as the target the rest as the predictors. Please determine the best combination among the others. Please do not forget to provide test-train comparison.**

```
#****linear regression******************
from sklearn import linear_model
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

#features -target
iris = datasets.load_iris()  # extract the dataset into iris
X = iris.data  # the data part will be here
y = iris.target
print(X)  # 4feature

#train-test
# Split the data into training/testing sets
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y,
random_state=42)

# Create linear regression object
regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
#output: [-0.11633589 -0.02686919  0.22653458  0.62162973]

# The mean squared error
print("Mean squared error: %.2f" % mean_squared_error(y_test,y_pred))

# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % r2_score(y_test, y_pred))


#LR Score, is the model overfit?
print("Training set score: {:.2f}".format(regr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(regr.score(X_test, y_test)))
```

Output:

```
Coefficients:
 [-0.11633589 -0.02686919  0.22653458  0.62162973]
Mean squared error: 0.06
Variance score: 0.91
Training set score: 0.94
Test set score: 0.91
```

According to MSE and variance score our LR is excellent. When we compare the LR Score of train and test dataset, they are close to each other which shows the model is not overfitted.

4. **Please apply the same procedure with Ridge and Lasso methods to the best dual combination in the question above with a slight mention about L1 and L2 regularization. Please briefly explain what these methods are sensitive to.**

> Technically the Lasso model is optimizing the objective function as the Elastic Net with L1_ratio=1.0 (no L2 penalty).

> Ridge: Linear least squares with L2 regularization. This model solves a regression model where the loss function is the linear least squares function and regularization is given by the L2-norm**. (from scikitlearn website)**

> I just copied the best Ridge and Lasso solution into pdf but the py file includes all of them with different alpha values and max_iteration.

```
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso

regr = linear_model.LinearRegression()
regr.fit(X_train, y_train)
y_pred = regr.predict(X_test)
print('LR Coefficients: \n', regr.coef_)
#LR Score, is the model overfit?
print("Training set score: {:.2f}".format(regr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(regr.score(X_test, y_test)))

ridge = Ridge().fit(X_train, y_train)  #alpha=1
y_pred=ridge.predict(X_test)
print('Ridge Coefficients: \n', ridge.coef_)
print("Training set score: {:.2f}".format(ridge.score(X_train, y_train)))
print("Test set score: {:.2f}".format(ridge.score(X_test, y_test)))


lasso00001 = Lasso(alpha=0.0001, max_iter=100000).fit(X_train, y_train)
print(" Lasso Training set score: {:.2f}".format(lasso00001.score(X_train,
y_train)))
print("Lasso Test set score: {:.2f}".format(lasso00001.score(X_test,
y_test)))
print('Lasso Coefficients: \n', lasso00001.coef_)
print("Number of features used: {}".format(np.sum(lasso00001.coef_ != 0)))
```

Output:

```
LR Coefficients:
 [-0.11633589 -0.02686919  0.22653458  0.62162973]
Training set score: 0.94
Test set score: 0.91
Ridge Coefficients:
 [-0.11902905 -0.01643189  0.26718254  0.52775965]
Training set score: 0.94
Test set score: 0.91
 Lasso Training set score: 0.94
Lasso Test set score: 0.91
Lasso Coefficients:
 [-0.11640477 -0.02597045  0.22757717  0.61938761]
Number of features used: 4
```

5. **Please run the code for iris dataset for three classification algorithms. Then determine the best one with explaining why you think this is the best one.**

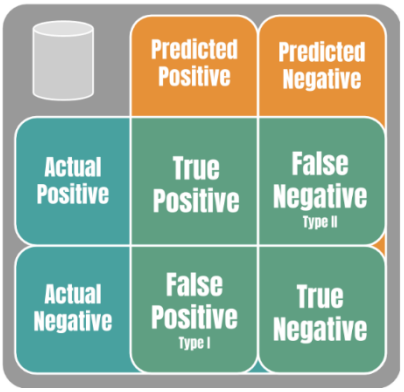
```
#Classification:
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier

X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X,y, stratify=y,
random_state=42)

#Decision Tree:
tree = DecisionTreeClassifier(max_depth=3)
tree.fit(X_train, y_train)
y_pred=tree.predict(X_test)
print("Decision tree accuracy score: ", accuracy_score(y_test, y_pred))
print("Accuracy on training set: {:.3f}".format(tree.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))

#SVC:
svc=svm.SVC()
svc.fit(X_train,y_train)
y_pred=svc.predict(X_test)
print("Decision tree accuracy score: ", accuracy_score(y_test, y_pred))
print("Accuracy on training set: {:.3f}".format(svc.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(svc.score(X_test, y_test)))

#RandomForest:
rfm=RandomForestClassifier()
rfm.fit(X_train,y_train)
y_pred=rfm.predict(X_test)
print("Decision tree accuracy score: ", accuracy_score(y_test, y_pred))
print("Accuracy on training set: {:.3f}".format(rfm.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(rfm.score(X_test, y_test)))
```

Output:

```
Decision tree accuracy score:  0.9210526315789473
Accuracy on training set: 0.982
Accuracy on test set: 0.921
SVC accuracy score:  1.0
Accuracy on training set: 0.982
Accuracy on test set: 1.000
Random Forest accuracy score:  0.9473684210526315
Accuracy on training set: 1.000
Accuracy on test set: 0.947
```

I select decision tree as classifier because there are obviously errors on the others when we consider the accuracy scores.

6. **What's the difference between Type I and Type II error? Explain your results with the outputs you obtained in the previous questions? (Hint: These errors are also known as false positive/negative).**



Error Type 1(FN): a false positive result states the condition when the condition is none.

Error Type 2(FP): a false negative result does not state the condition when the condition exists .

Accuracy is the number of correct predictions divided by the total number of predictions made.
When we look at previous question, SVC's accuracy score on test dataset is 1.0. It means the model assumes that all predictions are positive while some of actual values are negative on test dataset. It's Type 1 (False Positive) Error. It is same for Random forest, when we look at accuracy score of training dataset. It shows us our train and test data differ from each other or they are split as imbalanced. When we consider the difference between accuracy score of train and test dataset, SVC and Random Forest models look like overfitted. Methods listed Q8 should be used to solve this problem.

https://machinelearningmastery.com/classification-accuracy-is-not-enough-more-performance-measures-you-can-use/

7. **What are ensemble methods? What are they crucial and widely used? Provide a concrete example where ensemble techniques might be useful.**

Ensemble methods are supervised learning methods that combines several base models to generate one optimal predictive model.

There are three popular methods:

1. **Bagging** (**B**ootstrapping and **Agg**regation): This method is building multiple models (generally same type) from different subsamples of the training dataset. It aims to **reduce variance**. So, when we have high variance linear regression models with same train dataset and we are sure there is no overfitting, we can try bagging them to reduce variance

2. **Boosting:** This method refers to a set of algorithms which able to convert week models to strong models. It builds multiple models (generally same type) each of them learns to fix the prediction errors of a prior model in the chain. It aims to **reduce bias**. So, when we have Decision tree models on same train dataset with high bias and we are sure there is

no underfitting, we can try boosting. For example, Random forest creates several trees and calculates the best model for the dataset.

3. **Stacking:** This method combines multiple regression or classification over a meta-classification or regression method. The base models are trained based on training dataset, afterwards the meta-model is trained on the outputs of the base level model as features. It aims to **improve predictions.**

8. We have already mentioned that it is important to have a balanced dataset for developing a good model. How would you handle an imbalanced dataset?

Imbalanced dataset is generally a problem on classification where the classes have not equal proportion of data. For example, when you have a model train with gender data, if 90% of your train data is woman, your accuracy will be 90% but your model is garbage because of imbalanced dataset. For handling this problem, first we can try to gather more data sample. Secondly, we should try to change our performance metric. Confusion matrix, F1-score(harmonic mean of precision and recall), specificity (how many selected instances are relevant), sensitivity(how many relevant instances are selected) are more preferable than accuracy. Thirdly, we can try to resampling: under-sampling (when data size is sufficient, reduce the size of the larger class) and over-sampling (when data is insufficient, increase the size of smaller class by repetition, bootstrapping or SMOTE). Before over-sampling, cross validation should be implemented for avoiding overfitting.

9. What is overfitting? How do you ensure you're not overfitting with a model? (Hint: You can get help from this link: https://www.quora.com/How-can-I-avoid-overfitting)

Overfitting is basically explained as memorization. It is a modeling error which is caused by fitting on all data including noise. If the machine learning algorithm is too flexible or complex -too many features and etc..- it can end up memorizing the noise instead of finding the pattern in the data. So, this overfit model will make predictions based on noise. When there is huge difference between accuracy of train and test data, we can easily say our model is overfit!

How to prevent?

Try to reduce complexity and variance. Find the optimal balance between bias and variance!

- **Cross Validation:** Use the training data to generate different mini train-test splits and use these split parts to train the model. (A good method: k-fold cross validation, random selection)

- **Train with more data:** just give one more shot to your model if it is possible. 😊

- Ensembling: combine predictions from different models. (methods explained on Q7.)

-Try with simplier model: If tried model is over-capacity, this method can be beneficial.

- Early stopping: try to predict with less iteration before model start to memorize.

- Pruning: If the model is CART, pruning is a good idea to avoid overfitting.

- Dimension reduction: use PCA or LDA to reduce complexity of data.(reduce features)

- **penalize model complexity:** For LR or SVM (models that use weights for features), regularization(What is it? Basically, it keeps all features but reduce coefficient/weight of some features.). For unsupervised learning methods, Bayesian Nonparametric Prior probabilities.