HumeWork #4

Problem1 for two random variables X,Y, Let $\mu X, \mu Y$ and $\sigma X, \sigma Y$ denote their means and standard deviations.

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma X\ \sigma Y} = \frac{E[(X-\mu X)(Y-\mu Y)]}{\sigma X\ \sigma Y}$$

Let $Y_1, Y_2, \dots, Y_B$ be identically distributed random variables with pairwise correlation equal to $\rho > 0$. In other words, for each i and j with $i \neq j$, we have $\rho(Y_i, Y_j) = \rho > 0$

Show that following indentity holds:

$$var\left(\frac{Y_1 + Y_2 + \dots + Y_B}{B}\right) = \rho\sigma^2 + (1-\rho)\frac{\sigma^2}{B}$$

hint

$$var(X_1 + X_2 + \dots + X_B) = \sum_{i=1}^{B}\sum_{j=1}^{B} cov(x_i, x_j)$$

what is this question asking? Its asking to make left side equal to the right side

From the hint we get $i=j$ and $i \neq j$, the two cases.

So will do the steps of splitting by the two cases first two case

$i = j$

$cov(y_i, y_i) = var(y_i) = \sigma^2$

$i \neq j$

$cov(y_i, y_j) = p(y_i, y_j) * \sigma_y * \sigma_y = p * \sigma^2$

with those lets just plug

$var(y_1 + y_2 + \dots y_n) = \sum_{i=1}^{n} \sigma^2 + \sum_{i=j}^{n} P * \sigma^2$

$var(y_1 + y_2 + \dots + y_n) = B \times \sigma^2 + B * (B - 1) * P * \sigma^2$

$\Rightarrow \underline{B \times \sigma^2 + B^2 - m * p * \sigma^2}$

↙ factored out^{n}

$var(y_1 + y_2 + \dots + y_n) = \sigma^2 * \left(\frac{1}{B} + \frac{(B-1)}{B} \times P\right)$

$var(y + y_2 + \dots y_n) = \sigma^2 * \left(\frac{1}{B} + \frac{(B-1)}{B} \times P\right)$

rerrange

$var\left(\frac{y_1 + y_2 + \dots + y_n}{B}\right) = p * \sigma^2 + (1-p)\frac{\sigma^2}{B}$

1) b) Explain why identdy (1) is relevant to the Random Forest methods

Indentily (1) is relevant to the random forest methods because they are build on a random dicision tree, which helps with variance of the random forest calculations and classfications. Basically importance/relevance is the variance part from the indentily (1). The correlation of the different trees that get build under random Forrest prevent it from being to correlated to one another. Basically lowering the variance and covariance between two features in the calculation. In conclusion, the identily important due to it variance aspect of single feature, to feature n, and low reduction from random forests.

ieor_142_hw4_starter_code (2)

November 7, 2023

1 IEOR 142 HW4 Starter Code — Fall 2023

[1]:

```
# Dependencies
import numpy as np
import pandas as pd

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
# TODO: Put all dependencies here
```

[4]:

```
# Load in data

train_data = pd.read_csv('letters_train.csv', index_col = 0) # The index was␣
 ↪added to fix the index mistmatch from Unnamed:0 Column
test_data  = pd.read_csv('letters_test.csv', index_col = 0) # The index was␣
 ↪added to fix the index mistmatch from Unnamed:0 Column

#Would of droppe dthe index column, but not sure if its need into the future.
#train_data.head()
#test_data.head()
# TODO: Load in data (after analyzing the dataset, delete any ouputs such as df.
 ↪inf(), df.head(), et).
# this cell should not output anything
```

2 Question 2 (25 points)

[5]:

```
# TODO: Create new variable here

# BTW I did it in terms of 0 and 1 to make it easy to code, and one of the␣
 ↪answers on Ed was Endorsed
# for this to do it in terms of 0 and 1s.
```

```
train_data['isB'] = (train_data['letter'] == 'B').astype(int) # --> by default␣
  ↪it makes True or False, but I want number
test_data['isB'] = (test_data['letter'] == 'B').astype(int)# --> by default it␣
  ↪makes True or False, but I want number

# Here is the way the thing wanted us to do, you can do it doing this way but␣
  ↪its too much work,
# becuase you would need use dummies either way

# Train_data['isB'] = train_data['letter'].apply(lambda x: 'Yes' if x == 'B'␣
  ↪else 'No')
# Test_data['isB'] = test_data['letter'].apply(lambda x: 'Yes' if x == 'B' else␣
  ↪'No')
```

[6]:

```
# TODO: Split into X and y
# Splitting the Data into X and Y and dropping the columns, can also do the␣
  ↪split. But following lab.
X_train = train_data.drop(columns = ['letter', 'isB'])
X_test = test_data.drop(columns = ['letter', 'isB'])

y_train = train_data['isB']
y_test = test_data['isB']
```

2.0.1 Part A: Baseline Model (3 points)

[10]:

```
# Q1A code

# BaseLine model so we can make the value jsut simple 0.
baseLine_model = 0

# Btw doing .mean() because people used count and you get weird 9992 value, but␣
  ↪we need percentage.
baseline_1_acc = (y_test == baseLine_model).mean()

# TODO: calcuate baseline accuracy
print(f'Baseline Test Accuracy: {baseline_1_acc:.4f}')
```

```
Baseline Test Accuracy: 0.7743
```

2.0.2 Part B: Logistic Regression (5 points)

[13]:

```
# Q1B code

# TODO: For all questions: Create and train model, then make predictions, then␣
  ↪calculate accuracy
# Started using Model_1b, and stuff because colloberatory kept overwriting␣
  ↪variables, and would make the code wrong,
```

```python
# so giving them model1b name for each of the bottom provided text.
# Create the model first:
model_1b = LogisticRegression(max_iter = 10000) # btw this number can be 100 or 1000 and you still get 0.9401
model_1b.fit(X_train, y_train)

# Creating the predictions as instructions say
y_prob_1b = model_1b.predict_proba(X_test) # this was weird to write cause its usually jsut .predict
y_pred_1b = [1 if pred > 0.5 else 0 for pred in y_prob_1b[:,1]] # code from lab

# Then accuracy
model_1b_acc =  accuracy_score(y_test, y_pred_1b) # TODO: calcuate logistic accuracy accuracy
print(f'Logistic Regression Test Accuracy: {model_1b_acc:.4f}')
```

```
Logistic Regression Test Accuracy: 0.9401
```

2.0.3 Part C: AUC (2 point)

[15]:

```python
# Q1C code
model_1b_auc = roc_auc_score(y_test, y_prob_1b[:,1]) # TODO: Calculate logistic AUC
print(f'Logistic Regression Test AUC: {model_1b_auc:.4f}')
```

```
Logistic Regression Test AUC: 0.9785
```

2.0.4 Part D: Cross-validated CART (5 points)

Written Answer: I did the graph, but the instructions say to not have any of that stuff, so there seem to be best_ccp_alpha value in the code, which gives the value of 0.0010 for alpha. And I used the np.linespace values from previous lab and Homework values. The Random_state is 2023 indicated by thet instructions. Used CV of 5, you can also use anything higher you still get the same value, so might as well just use 5. We get the accuracy of 0.9401 which is pretty good, thats about 94% accurate at predicting on the test set.

[18]:

```python
# Q1D Code
params_1d = {'ccp_alpha': np.linspace(0, 0.1, 100)}
model_1d = DecisionTreeClassifier(random_state = 2023)
model_1d_cv = GridSearchCV(model_1d, params_1d, cv = 5, scoring = 'accuracy',verbose = False)
model_1d_cv.fit(X_train, y_train)
model_1d_best_ccp_alpha = model_1d_cv.best_params_['ccp_alpha']
y_prob_1d = model_1d_cv.best_estimator_.predict_proba(X_test)
y_pred_1d = [1 if pred > 0.5 else 0 for pred in y_prob_1d[:,1]]
```

```
model_1d_acc = accuracy_score(y_test, y_pred_1d) # TODO: calculate best CV␣
  ↪accuracy
model_1d_best_ccp_alpha =  model_1d_best_ccp_alpha # TODO: get best ccp_alpha
print(f'CV CART Test Accuracy: {model_1d_acc:.4f}')
print(f'Best ccp_alpha: {model_1d_best_ccp_alpha:.4f}')
```

```
CV CART Test Accuracy: 0.9401
Best ccp_alpha: 0.0010
```

2.0.5 Part E: Random Forest (5 points)

[21]:

```
# Q1E Code
model_1e = RandomForestClassifier(random_state = 2023)
model_1e.fit(X_train, y_train)
y_prob_1e = model_1e.predict_proba(X_test)
y_pred_1e = [1 if pred > 0.5 else 0 for pred in y_prob_1e[:,1]]

model_1e_acc = accuracy_score(y_test, y_pred_1e) # TODO: calculate random␣
  ↪forest accuracy
print(f'Random Forest Test Accuracy: {model_1e_acc:.4f}')
```

```
Random Forest Test Accuracy: 0.9840
```

2.0.6 Part F: Performance Comparison (5 points)

Written Answer: The accuracy comparion between the model goes from the lowest to highest, Baseline, lowest with 77% accuracy, then Regression anad Cart tied at 94% and Random Forest with the highest accuracy at 98%. So the highest accracy model is Random Forest. For the answer of whether interpretability or accuracy matter, it depends on the case, if you just want to be able to identify something, you would go with the highest accuracy but if its something like insurance, where missclassifying can cause a lot of money, then you would want interpredability over accracy there. Here I would pick regression or Cart because they both got the same accuracy, they may signify the models aren't being overfitted onto anything.

[27]:

```
# Q1F Code

data = [
    ["Baseline", baseline_1_acc],
    ["Regression", model_1b_acc],
    ["CART", model_1d_acc],
    ['Random Forest', model_1e_acc]
]

# Create the pandas DataFrame
df = pd.DataFrame(data, columns=['Model Type', 'Accuracy '])

# Print the dataframe.
df
```

```
# got this from geeksforgeeks
# https://www.geeksforgeeks.org/different-ways-to-create-pandas-dataframe/
# TODO: create df to compare performance
```

[27]:

```
      Model Type  Accuracy
0       Baseline  0.774332
1     Regression  0.940107
2           CART  0.940107
3  Random Forest  0.983957
```

---

3 Question 3 (50 points)

[29]:

```
# TODO: Redefine target y
y_train_1 = train_data['letter']
y_test_1  = test_data['letter']
```

3.0.1 Part A: Baseline Model (5 points)

[31]:

```
# Q2A
base_model_2 = y_train_1.value_counts().index[0]   # this is to predict one␣
 ↪letter, which is p in thise case                                              ␣
 ↪
baseline_2_acc = (y_test_1 == base_model_2).mean() # TODO: calculate baseline␣
 ↪accuracy
print(f'Baseline Test Accuracy: {baseline_2_acc:.4f}')
```

```
Baseline Test Accuracy: 0.2439
```

3.0.2 Part B: LDA (8 points)

[32]:

```
# Q2B code

model_2b = LDA()
model_2b.fit(X_train, y_train_1)
y_pred_2b = model_2b.predict(X_test)

model_2b_acc = accuracy_score(y_test_1, y_pred_2b) # Calculate LDA accuracy
print(f'LDA Test Accuracy: {model_2b_acc:.4f}')
```

```
LDA Test Accuracy: 0.9102
```

3.0.3 Part C: Cross-validated CART (10 points)

Written Answer: Same way as above question 1c. I did the graph, but the instructions say to not have any of that stuff, so there seem to be best_ccp_alpha value in the code, which gives the value of 0.0010 for alpha. And I used the np.linespace values from previous lab and Homework values. The Random_state is 2023 indicated by thet instructions. Used CV of 5, you can also use anything higher you still get the same value, so might as well just use 5. We get the accuracy of 0.9283 which is pretty good, thats about 9283% accurate at predicting on the test set. Also I used scoring as accruacy, which was used last time on homework and lab.

[45]:

```
# Q2C Code
params_2 = {'ccp_alpha': np.linspace(0, 0.1, 100)}
model_2c = DecisionTreeClassifier(random_state = 2023)
model_2c_cv = GridSearchCV(model_2c, params_2, cv = 10, scoring = 'accuracy',
                           verbose = False)
model_2c_cv.fit(X_train, y_train_1) # can be either accracy or r2


y_pred_2c = model_2c_cv.best_estimator_.predict(X_test)

model_2c_acc = accuracy_score(y_test_1, y_pred_2c) # TODO: calculate CV CART␣
 ↪accuracy
print(f'CART Test Accuracy: {model_2c_acc:.4f}')
```

```
CART Test Accuracy: 0.9283
```

3.0.4 Part D: Vanilla Bagging (8 points)

[40]:

```
# Q2D
max_features_length = len(X_train.columns)

model_2d = RandomForestClassifier(max_features = max_features_length,
                                  random_state = 2023)
model_2d.fit(X_train, y_train_1)
y_pred_2d = model_2d.predict(X_test)


model_2d_acc = accuracy_score(y_test_1, y_pred_2d) # TODO: Calculate vanilla␣
 ↪bagging accuracy
print(f'No CV Random Forest Test Accuracy: {model_2d_acc:.4f}')
```

```
No CV Random Forest Test Accuracy: 0.9476
```

3.0.5 Part E: Cross-validated Random Forest (10 points)

Written Answer: For cross validation I just picked 5, you can pick anything about 5 and it won't change the result too much. For length I picked of features I picked all the features, you get 16 starting at 1. so we do +1 to the range to prevent it from starting 0 and breaking the code all together. The test acruacy is 0.9765, which is pretty good.

[44]:

```
# Q2E

# I had to +1 to prevent the code from breaking. and start np.range at 1 and␣
 ↪not 0.
```

```
range = len(X_train.columns) + 1 # I am not sure why you do + 1 someone jsut␣
 ↪told me to do it on discord idk.
dictioanry = dict(max_features = np.arange(1, param_range)) # it was to prevent␣
 ↪the offeset, and you get 16. Cant't


model_2e = RandomForestClassifier(random_state = 2023)
model_2e_cv = GridSearchCV(model_2e, dictioanry, cv = 5, scoring =␣
 ↪'accuracy',verbose = False)
model_2e_cv.fit(X_train, y_train_1)
y_pred_2e = model_2e_cv.best_estimator_.predict(X_test)
model_2e_acc = accuracy_score(y_test_1, y_pred_2e)

model_2e_acc = accuracy_score(y_test_1, y_pred_2e) # TODO: Calculate RF accuracy
print(f'CV Random Forest Test Accuracy: {model_2e_acc:.4f}')
```

```
CV Random Forest Test Accuracy: 0.9765
```

3.0.6 Part F: Gradient Boosting Classifier (9 points)

[90]:

```
# Q2F
model_2f = GradientBoostingClassifier(n_estimators = 200,
                                      max_leaf_nodes = 10,
                                      random_state = 2023)
model_2f.fit(X_train, y_train_2)
y_pred_2f = model_2f.predict(X_test)

model_2f_acc = accuracy_score(y_test_2, y_pred_2f) # Calculate boosting accuracy
print(f'GBC Test Accuracy: {model_2f_acc:.4f}')
```

```
GBC Test Accuracy: 0.9701
```

[ ]: