مقدمه:

در این پروژه می‌خواهیم با استفاده از چند الگوریتم طبقه بندی با استفاده از یک دیتا ست احتمال برگشتن یا بر نگشتن مشتریان را بررسی کنیم که برای این کار نیاز به پیاده‌سازی الگوریتم ها با کتاب خانه و پیش پردازش روی داده‌ها داریم

پیش پردازش:

برای اینکه بتوانیم داده‌ها را برای الگوریتم های طبقه بندی استفاده کنیم نیاز به پیش پردازش روی آن‌ها داریم برای ستون‌های مختلف پیش پردازش های انجام شده را توضیح می دهیم:

customer ID:

این ستون شامل یک ایدی برای هر مشتری است. این اعداد همگی با هم فرق دارند به همین دلیل گین زیادی را به ما می‌دهد جون می‌تواند به راحتی آن را جدا کرد ولی این گین بی‌فایده است چون هیچ دو عددی با هم برابر نیستند و نمی‌تواند دقت الگوریتم ما را برای جدا سازی بیشتر کند به همین دلیل می‌تواند این ستون را حذف کرد

که در آن پروژه من این ستون را حذف کرده ام:

```
self.DataFrame = pd.read_csv(filepath, index_col=0).drop('Customer ID', axis=1)
```

Total quality & Total Price:

این دو ستون دارای مجموعه‌ای از اعداد هستند. برای این دسته از داده‌ها یک سری داده ی منفی وجود دارد که احتمالاً غلط در دیتا ست یا نبودن عدد است. این اعداد و ردیف آن را می‌توان حذف کرد و یا همین گونه که هستند در نظر گرفت که در آنجا من آن‌ها را حذف کرده‌ام

سپس می‌توان این داده‌ها را اسکیل کرد. می‌دانیم اسکیل کردن داده‌ها باعث می‌شود رنج آن‌ها کم شود این کار در الگوریتم هایی مثل K-NN می‌تواند تأثیر داشته باشد چرا که پیدا کردن تابع فاصله راحت‌تر است ولی برای الگوریتم های درخت فرقی ندارد

چرا که مقدار treshhold آن نیز کم می‌شود و داده‌ها همچنان با آن ستون مانند قبل جدا می شوند.
در اینجا من داده‌ها را اسکیل کردم چرا که با آزمون و خطا دیده شد می‌تواند دقت الگوریتم ها را در مجموع بالا بیاورد

```
def scale_data(self):
    scaler = StandardScaler()
    new_data = list(zip(self.DataFrame["Total Quantity"], self.DataFrame["Total Price"]))
    scaler.fit(new_data)
    new_data = scaler.transform(new_data)
    self.DataFrame["Total Quantity"] = new_data[:, 0]
    self.DataFrame["Total Price"] = new_data[:, 1]
```

Country:

این ستون شامل کشور های مشتریان است. این ستون دارای اسم کشور ها است ولی ما برای الگوریتم های طبقه بندی نیاز داریم تا آن‌ها را به اعداد تبدیل کنیم تا بتوانیم آن‌ها را با یک Treshhold از هم جدا کنیم.
برای این کار چند روش می‌تواند انجام داد Label کردن آن هاست در این روش برای هر کشور یک عدد داده می‌شود و ما به تعداد کشور ها عدد داریم و آن‌هایی که کشور برابر دارند عدد یکسان دارند پس یعنی به هر کشور یک لیبل عدد داده‌ایم این کار در این پروژه می‌شود انجام داد چرا که ما نیاز داریم تا کشور های شبیه را از هم جدا کنیم که با این کار قابل انجام است
روش دیگر استفاده از one hot encoder است در این روش برای هر کشور یک ستون به داده‌های ما اضافه می‌شود در این صورت برای هر کشور ستون مربوط به آن ۱ می‌شود در این حالت برای دیتا ست ما ۳۷ ستون اضافه می‌شود
فرق دو روش این است که در لیبل کردن این مقادیر مدل ما ممکن است به اشتباه فکر کند این داده‌ها یک ترتیب خاصی دارند چرا که آن‌ها به اعداد ربط داده شده‌اند که این‌طور نیست و در این صورت ممکن است دقت پایین بیاید
در اینجا هر دو روش را امتحان کرده‌ام و دقت آن‌ها تقریباً برابر است

Date:

برای این ستون باید تاریخ را به یک مقدار عددی تبدیل کنیم. برای این کار می‌توانیم از روز در ماه یا ماه یا سال آن یا ترکیبی از آن‌ها استفاده کنیم. من در این پروژه از ترکیب روز و ماه آن استفاده کرده‌ام به این صورت که 11-05 درواقع همان 1105 می شود.

```
def date_get_day(self):
    self.DataFrame.Date = self.DataFrame.Date.str.split('-').str[1] + self.DataFrame.Date.str.split('-').str[2]
    self.DataFrame.Date = pd.to_numeric(self.DataFrame.Date)
```

IS Back:

این ستون درواقع خروجی ما است که می‌خواهیم بدانیم آیا مشتریان دوباره بر می‌گردند یا خیر. که این ستون را به عنوان خروجی از بقیه ی ستون‌ها جدا می کنیم.

```
def split_labels(self, label_name="Is Back"):
    desired_width = 320
    pd.set_option('display.width', desired_width)
    pd.set_option('display.max_columns', 10)

    self.country_encode()
    self.date_get_day()

    # print(self.DataFrame[self.DataFrame['Total Quantity'].map(len)])
    self.DataFrame = self.DataFrame.drop(self.DataFrame[(self.DataFrame['Total Quantity'] < 0)].index)
    self.DataFrame = self.DataFrame.drop(self.DataFrame[(self.DataFrame['Total Price'] < 0)].index)

    self.scale_data()
    x = self.DataFrame.drop(label_name, axis=1)
    y = self.DataFrame[label_name]
    return x, y
```

و در آخر باید یک سری داده تست و ترین را از هم جدا کنیم تا بتوانیم با آن‌ها مدل هایمان را تست کنیم
در این پروژه over sampling هم انجام شده است تا تعداد کلاس‌ها برابر شود

```
def split_validation(self, validation_rate=0.4):
    x, y = self.split_labels()

    ros = RandomOverSampler(random_state=0)
    x, y = ros.fit_resample(x, y)

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=validation_rate)
    return x_train, x_test, y_train, y_test
```

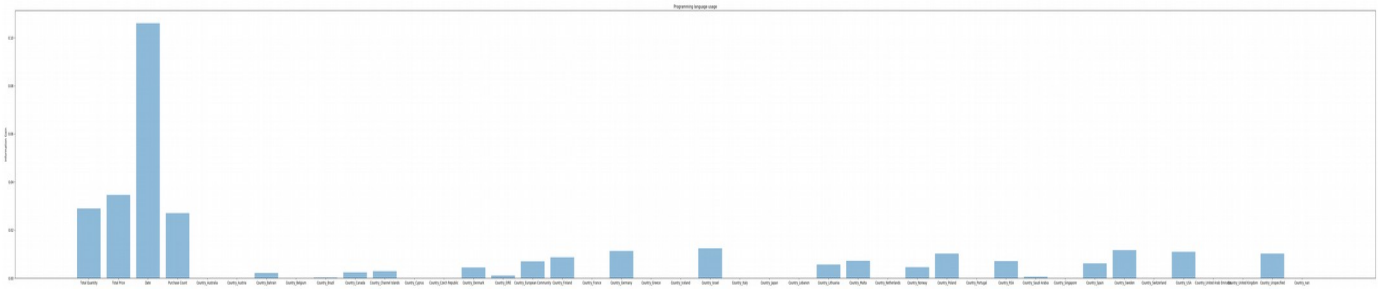
فاز صفر:

در این قسمت می‌خواهیم با استفاده از پیدا کردن information gain مربوط به مقادیر اولیه بهترین ویژگی برای جدا کردن داده‌های مختلف را پیدا کنیم برای این کار از کتاب خانه و داده‌های جدا شده در قسمت قبل استفاده می کنیم:

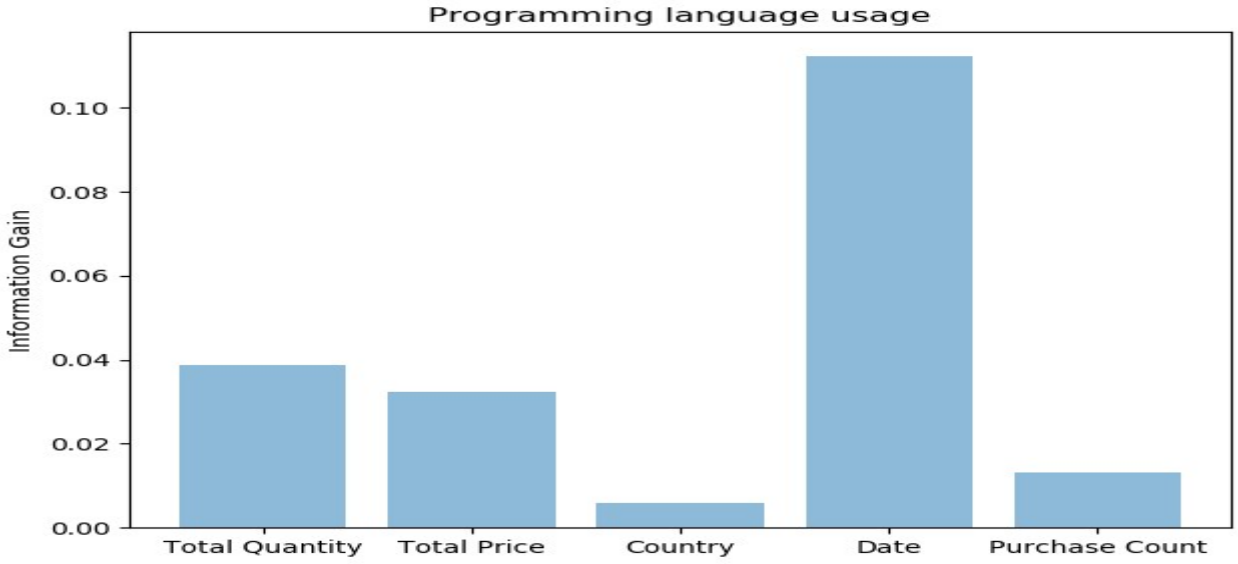
```
def information_gain(self):
    res = dict(zip(self.x_train.columns.values, mutual_info_classif(self.x_train, self.y_train)))
    print(res)
    objects = self.x_train.columns.values
    y_pos = np.arange(len(objects))
    performance = mutual_info_classif(self.x_train, self.y_train)

    plt.figure(figsize=(80, 10))
    plt.bar(y_pos, performance, align='center', alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.ylabel('Information Gain')
    plt.title('Programming language usage')
    plt.show()
```

و نمودار آن به شکل زیر می شود:

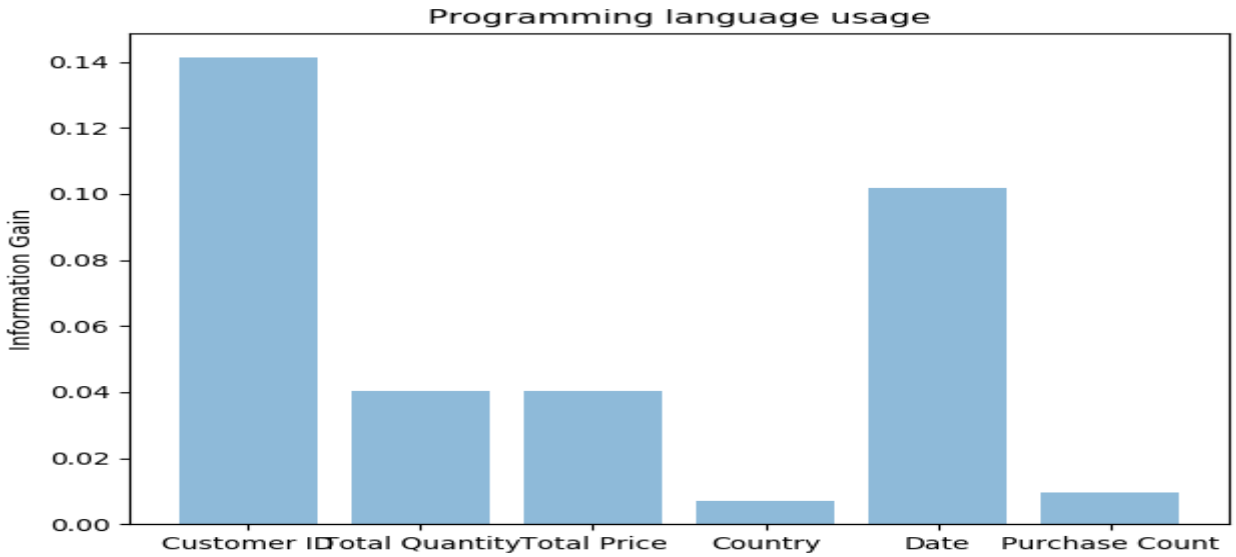
اگر کشور ها را به شکل one hot encode کنیم:



و اگر از لیبل کردن استفاده کنیم:



که همان‌طور که دیده می‌شود در درجه اول تاریخ قرار دارد چرا که این وزگی به خوبی می‌تواند مشتریان را از هم جدا کند زیرا اعداد تاریخ با یک دیگر متفاوت اند و در هر تاریخ می‌تواند فهمید که آیا مشتریان باز می‌گردند یا خیر. این مقدار لزوماً برای ما درست نیست چون اعداد آن با هم فرق دارند و نمی‌تواند جدا ساز خوبی برای ما باشد همانند customer id.



می‌بینیم که customer id هم گین بالایی دارد چرا که اعداد آن کاملاً با هم متفاوت است.
اعداد customer id تفاوت بیشتری دارند به همین دلیل گین بالاتری دارد
و همان‌طور که دیده می‌شود مقادیر price و quantity تقریباً برابر هستند چرا که پخشی آن‌ها تقریباً برابر است

فاز اول:

در این قسمت می‌خواهیم ۳ الگوریتم طبقه بندی را استفاده و تحلیل کنیم که آن‌ها را یک به یک بررسی می کنیم:

درخت تصمیم:

اولین الگوریتم درخت تصمیم است این الگوریتم را با عمق های مختلف امتحان می‌کنیم تا به جایی برسیم که دیگر overfit نداشته باشیم و به دقت مناسبی برسیم. برای پیاده‌سازی آن از کتابخانه استفاده می‌کنیم و عمق را به صورت hperparameter به آن می دهیم:

```
classifier = tree.DecisionTreeClassifier(max_depth=max_depth, random_state=0)
classifier.fit(self.x_train, self.y_train)

y_predict_train = classifier.predict(self.x_train)
y_predict = classifier.predict(self.x_test)
print("Decision Tree")
print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_trai
print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
print("\n")
```

برای مثال این الگوریتم با عمق ۱۰ داریم:

```
Decision Tree
For Train Predicts
              precision    recall  f1-score   support

          No       0.83      0.84      0.84      1730
         Yes       0.84      0.82      0.83      1680

    accuracy                           0.83      3410
   macro avg       0.83      0.83      0.83      3410
weighted avg       0.83      0.83      0.83      3410

Accuracy:  0.8322580645161290
For Test Predicts
              precision    recall  f1-score   support

          No       0.70      0.74      0.72      1112
         Yes       0.74      0.69      0.71      1162

    accuracy                           0.72      2274
   macro avg       0.72      0.72      0.72      2274
weighted avg       0.72      0.72      0.72      2274

Accuracy:  0.7163588390501319
```

نزدیک‌ترین همسایه:
این الگوریتم را نیز مانند قبلی با کتابخانه پیاده‌سازی می‌کنیم و تعداد نقاط همسایه را به عنوان hyperparameter در نظر می‌گیریم و داریم:

```
neigh = neighbors.KNeighborsClassifier(n_neighbors=k)
neigh.fit(self.x_train, self.y_train)

y_predict_train = neigh.predict(self.x_train)
y_predict = neigh.predict(self.x_test)

print("K Nearest Neighbor")
print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
print("\n")
```

و برای ۵ نقطه در همسایگی داریم:

```
K Nearest Neighbor
For Train Predicts
              precision    recall  f1-score   support

          No       0.76      0.82      0.79      1730
         Yes       0.80      0.73      0.76      1680

    accuracy                           0.78      3410
   macro avg       0.78      0.78      0.78      3410
weighted avg       0.78      0.78      0.78      3410

Accuracy:  0.7768328445747801
For Test Predicts
              precision    recall  f1-score   support

          No       0.64      0.70      0.67      1112
         Yes       0.69      0.63      0.66      1162

    accuracy                           0.66      2274
   macro avg       0.67      0.67      0.66      2274
weighted avg       0.67      0.66      0.66      2274

Accuracy:  0.6644678979771328
```

:Logistic Classifier
این الگوریتم را نیز مانند قبلی‌ها با استفاده از کتاب خانه پیاده‌سازی می‌کنیم با این تفاوت که دیگر به آن hyperparameter ای نداده ایم:

```python
def logistic_classifier(self):
    logistic = linear_model.LogisticRegression()
    logistic.fit(self.x_train, self.y_train)

    y_predict_train = logistic.predict(self.x_train)
    y_predict = logistic.predict(self.x_test)

    print("Logistic Classifier")
    print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
    print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
    print("\n")
```

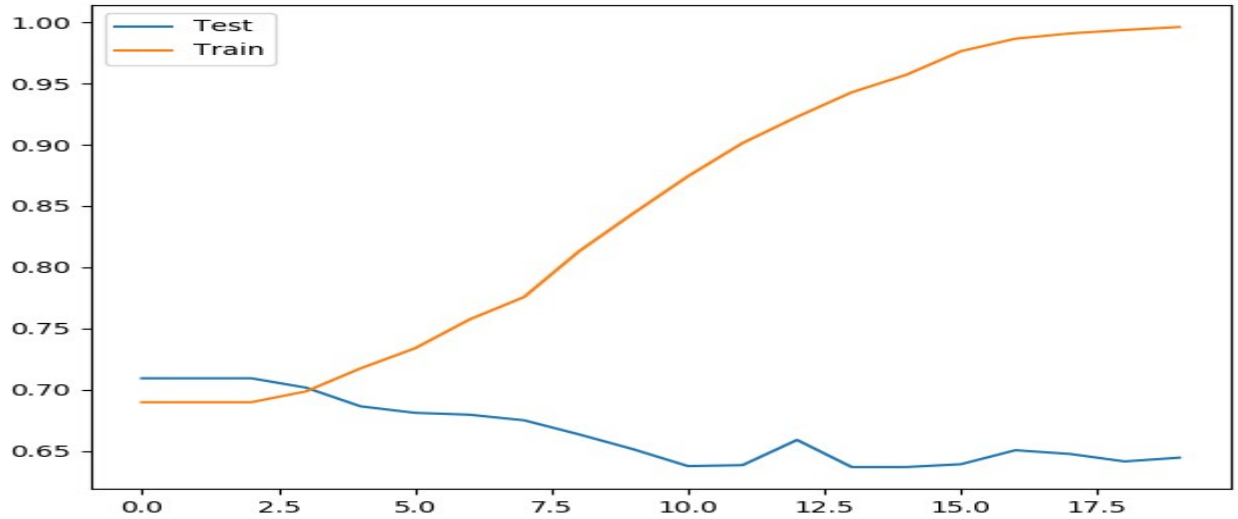
و بعد از اجرای آن داریم:

```
Logistic Classifier
For Train Predicts
              precision    recall  f1-score   support

          No       0.58      0.63      0.60      1730
         Yes       0.58      0.53      0.55      1680

    accuracy                           0.58      3410
   macro avg       0.58      0.58      0.58      3410
weighted avg       0.58      0.58      0.58      3410

Accuracy:  0.5780058651026393
For Test Predicts
              precision    recall  f1-score   support

          No       0.56      0.64      0.60      1112
         Yes       0.60      0.53      0.56      1162

    accuracy                           0.58      2274
   macro avg       0.58      0.58      0.58      2274
weighted avg       0.58      0.58      0.58      2274

Accuracy:  0.5822339489885664
```

حال می‌خواهیم دو الگوریتم درخت تصمیم و همسایگی را برای مفادیر مختلف hyperparameter های آن بررسی کنیم و نمودار های آن‌ها را رسم کنیم و overfit را بررسی کنیم
برای اینکار ابتدا درخت تصمیم را مانند شکل زیر درست می کنیم:

```
def decision_tree(self, max_depth):
    accuracy_history_train = []
    accuracy_history_test = []
    for i in range(max_depth):
        print("Epoch number: ", i)
        classifier = tree.DecisionTreeClassifier(max_depth=i + 1, random_state=0)
        classifier.fit(self.x_train, self.y_train)

        y_predict_train = classifier.predict(self.x_train)
        y_predict = classifier.predict(self.x_test)
        print("Decision Tree")
        print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
        print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
        print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
        print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
        print("\n")

        accuracy_history_test.append(sklearn.metrics.accuracy_score(self.y_test, y_predict))
        accuracy_history_train.append(sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
    self.plot_accuracy(accuracy_history_train, accuracy_history_test)
```

حال اگر این درخت را برای عمق ۱ تا ۲۰ رسم کنیم شکل آن به صورت زیر می شود:

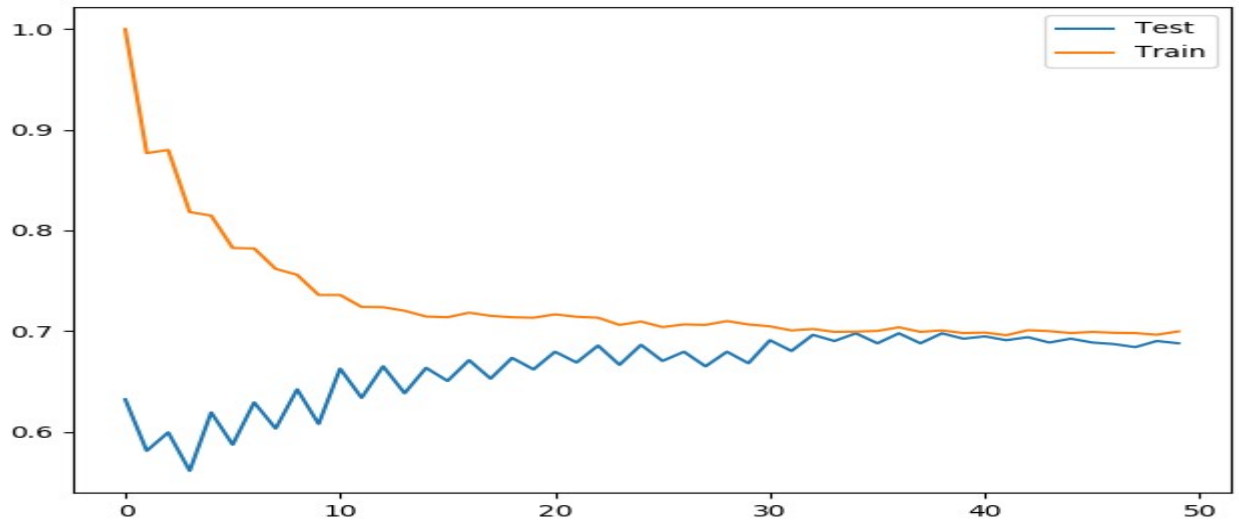


همان‌طور که میبینیم بعد از تقریباً عمق ۵ دقت آموزش بالا می‌رود ولی تست بالا نمی‌رود که به این معنی است بعد از آن overfit اتفاق افتاده و درخت نمی‌تواند داده‌های تست را درست جدا کند و فقط برای داده‌های آموزش کار می کند

حال می‌توانیم همین کار ها را برای همسایگی نیز انجام دهیم و داریم:

```python
def k_nearest_neighbors(self, k):
    accuracy_history_train = []
    accuracy_history_test = []
    for i in range(k):
        print("Epoch number: ", i)
        neigh = neighbors.KNeighborsClassifier(n_neighbors=i + 1)
        neigh.fit(self.x_train, self.y_train)

        y_predict_train = neigh.predict(self.x_train)
        y_predict = neigh.predict(self.x_test)

        print("K Nearest Neighbor")
        print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
        print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
        print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
        print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
        print("\n")

        accuracy_history_test.append(sklearn.metrics.accuracy_score(self.y_test, y_predict))
        accuracy_history_train.append(sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
    self.plot_accuracy(accuracy_history_train, accuracy_history_test)
```

و اگر آن را برای اعداد ۱ تا ۲۰ امتحان کنیم نمودار آن به شکل زیر می شود:



همان‌طور که می‌بینیم با در نظر گرفتن تعداد نقاط بیشتر دقت پایین‌تر می‌رود ولی برای داده‌ها تست نیز بهتر عمل می‌کند مه در جایی مثلاً ۴۰ این دو نزدیک به هم بوده و دقت بالا تری دارند

فاز دوم:

در این قسمت می‌خواهیم با استفاده از الگوریتم های طبقه بندی گروهی برگشتن یا برنگشتن مشتریان را پیش‌بینی کنیم
در قسمت اول از bagging استفاده می کنیم:
این روش با استفاده از تعداد n تا از یک الگوریتم و استفاده از نسبتی از تعداد کل فیچر ها خروجی را حدس می‌زند
ابتدا این کار را برای درخت تصمیم انجام می دهیم:

```
classifiers.bagging(tree.DecisionTreeClassifier(max_depth=4), 10)
```

و برای پیاده‌سازی آن داریم:

```
def bagging(self, estimator, number_of_estimator):
    bag = BaggingClassifier(base_estimator=estimator, n_estimators=number_of_estimator, max_features=0.5)
    bag.fit(self.x_train, self.y_train)

    y_predict_train = bag.predict(self.x_train)
    y_predict = bag.predict(self.x_test)

    print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
    print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
```

و بعد از اجرای آن داریم:

```
For Train Predicts
                precision    recall  f1-score   support

           No       0.74      0.68      0.71      2099
          Yes       0.70      0.76      0.73      2099

    accuracy                           0.72      4198
   macro avg       0.72      0.72      0.72      4198
weighted avg       0.72      0.72      0.72      4198

Accuracy:  0.7196283944735589
For Test Predicts
                precision    recall  f1-score   support

           No       0.53      0.63      0.58       423
          Yes       0.81      0.73      0.77       887

    accuracy                           0.70      1310
   macro avg       0.67      0.68      0.67      1310
weighted avg       0.72      0.70      0.71      1310

Accuracy:  0.7007633587786259
```

حال می‌توانیم همین کار ها را برای الگوریتم همسایگی نیز بدست اوریم:

```
classifiers.bagging(neighbors.KNeighborsClassifier(n_neighbors=40), 10)
```

و برای پیاده‌سازی ان:

```
def bagging(self, estimator, number_of_estimator):
    bag = BaggingClassifier(base_estimator=estimator, n_estimators=number_of_estimator, max_features=0.5)
    bag.fit(self.x_train, self.y_train)

    y_predict_train = bag.predict(self.x_train)
    y_predict = bag.predict(self.x_test)

    print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
    print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
```

و بعد از اجرای آن داریم:

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.75      0.68      0.71      2099
         Yes       0.71      0.77      0.74      2099

    accuracy                           0.73      4198
   macro avg       0.73      0.73      0.73      4198
weighted avg       0.73      0.73      0.73      4198

Accuracy:  0.7258218199142449
For Test Predicts
              precision    recall  f1-score   support

          No       0.53      0.61      0.57       423
         Yes       0.80      0.74      0.77       887

    accuracy                           0.70      1310
   macro avg       0.66      0.68      0.67      1310
weighted avg       0.71      0.70      0.70      1310

Accuracy:  0.6977099236641221
```

که همان‌طور که میبینیم دقت برای درخت تصمیم بهتر شده و این دوش برای الگوریتم اول جواب بهتری داده است

در قسمت بعد می‌خواهیم الگوریتم RandomForest را پیاده‌سازی کنیم و تأثیر پارامتر ها را روی آن ببینیم.
برای پیاده‌سازی آن داریم:

```
def random_forest(self, depth, estimators, bootstrap):
    clf = RandomForestClassifier(max_depth=depth, n_estimators=estimators, bootstrap=bootstrap,random_state=0)
    clf.fit(self.x_train, self.y_train)

    y_predict_train = clf.predict(self.x_train)
    y_predict = clf.predict(self.x_test)

    print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
    print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
```

همان‌طور که دیده می‌شود hyperparameter های مورد بررسی عمق درخت و تعداد پیش‌بینی کننده ها یا همان تعداد دخت ها و استفاده از bootstrap است حال این الگوریتم را با حالت‌های مختلف اجرا می‌کنیم و خروجی را می بینیم:

برای تعداد درخت داریم:

```
classifiers.random_forest(4, 10, True)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.75      0.62      0.68      2099
         Yes       0.68      0.79      0.73      2099

    accuracy                           0.71      4198
   macro avg       0.71      0.71      0.70      4198
weighted avg       0.71      0.71      0.70      4198

Accuracy:  0.7070033349213911
For Test Predicts
              precision    recall  f1-score   support

          No       0.56      0.56      0.56       423
         Yes       0.79      0.79      0.79       887

    accuracy                           0.72      1310
   macro avg       0.67      0.68      0.68      1310
weighted avg       0.72      0.72      0.72      1310

Accuracy:  0.7152671755725191
```

```
classifiers.random_forest(4, 50, True)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.75      0.62      0.68      2099
         Yes       0.68      0.79      0.73      2099

    accuracy                           0.71      4198
   macro avg       0.71      0.71      0.70      4198
weighted avg       0.71      0.71      0.70      4198

Accuracy:  0.7067651262505955
For Test Predicts
              precision    recall  f1-score   support

          No       0.55      0.57      0.56       423
         Yes       0.79      0.78      0.79       887

    accuracy                           0.71      1310
   macro avg       0.67      0.68      0.67      1310
weighted avg       0.72      0.71      0.71      1310

Accuracy:  0.7129770992366412
```

```
classifiers.random_forest(4, 500, True)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.75      0.62      0.68      2099
         Yes       0.68      0.79      0.73      2099

    accuracy                           0.71      4198
   macro avg       0.71      0.71      0.70      4198
weighted avg       0.71      0.71      0.70      4198

Accuracy:  0.7055740828966174
For Test Predicts
              precision    recall  f1-score   support

          No       0.55      0.57      0.56       423
         Yes       0.79      0.78      0.78       887

    accuracy                           0.71      1310
   macro avg       0.67      0.67      0.67      1310
weighted avg       0.71      0.71      0.71      1310

Accuracy:  0.7106870229007634
```

می‌بینیم دقت در دو حالت آخر در تست بیشتر از ترین شده است م تقریباً ثابت است که یعنی مدل با تعداد درخت بیشتر به درستی تشخیص می‌دهد و از یک جا به بعد این افزایش تأثیری نداشته است

حال برای عمق درخت داریم:

```
classifiers.random_forest(4, 100, True)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.75      0.62      0.68      2099
         Yes       0.68      0.79      0.73      2099

    accuracy                           0.71      4198
   macro avg       0.71      0.71      0.70      4198
weighted avg       0.71      0.71      0.70      4198

Accuracy:  0.7070033349213911
For Test Predicts
              precision    recall  f1-score   support

          No       0.56      0.56      0.56       423
         Yes       0.79      0.79      0.79       887

    accuracy                           0.72      1310
   macro avg       0.67      0.68      0.68      1310
weighted avg       0.72      0.72      0.72      1310

Accuracy:  0.7152671755725191
```

```
classifiers.random_forest(20, 100, True)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       1.00      1.00      1.00      2099
         Yes       1.00      1.00      1.00      2099

    accuracy                           1.00      4198
   macro avg       1.00      1.00      1.00      4198
weighted avg       1.00      1.00      1.00      4198

Accuracy:  1.0
For Test Predicts
              precision    recall  f1-score   support

          No       0.55      0.50      0.53       423
         Yes       0.77      0.81      0.79       887

    accuracy                           0.71      1310
   macro avg       0.66      0.65      0.66      1310
weighted avg       0.70      0.71      0.70      1310

Accuracy:  0.7076335877862595
```

در حالت دوم می‌دانیم درخت ما کاملاً overfit است یعنی برای داده‌های آموزش کامل درست است در این صورت در داده‌های تست باید دقت پایین باشد ولی می‌بینیم حتی در این حالت نیز دقت برای تست پایین نیامده است و مدل توانسته مقداری از overfit را باز هم جبران کند در حالی که اگر فقط ۱ درخت داشتیم دقت تست همانطور که در فاز قبل بود نزدیک به ۰ می‌شد

و برای bootstraping داریم:

```
classifiers.random_forest(4, 100, True)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.75      0.62      0.68      2099
         Yes       0.68      0.79      0.73      2099

    accuracy                           0.71      4198
   macro avg       0.71      0.71      0.70      4198
weighted avg       0.71      0.71      0.70      4198

Accuracy:  0.7070033349213911
For Test Predicts
              precision    recall  f1-score   support

          No       0.56      0.56      0.56       423
         Yes       0.79      0.79      0.79       887

    accuracy                           0.72      1310
   macro avg       0.67      0.68      0.68      1310
weighted avg       0.72      0.72      0.72      1310

Accuracy:  0.7152671755725191
```

```
classifiers.random_forest(4, 100, False)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.75      0.63      0.68      2099
         Yes       0.68      0.79      0.73      2099

    accuracy                           0.71      4198
   macro avg       0.72      0.71      0.71      4198
weighted avg       0.72      0.71      0.71      4198

Accuracy:  0.709623630300143
For Test Predicts
              precision    recall  f1-score   support

          No       0.56      0.57      0.56       423
         Yes       0.79      0.78      0.79       887

    accuracy                           0.71      1310
   macro avg       0.67      0.68      0.67      1310
weighted avg       0.72      0.71      0.71      1310

Accuracy:  0.7137404580152672
```

حال می‌خواهیم تأثیر bagging را بر overfit بررسی کنیم برای این کار این روش را یک بار روی یک درخت و یک بار روی یک همسایگی با overfit بالا امتحان می‌کنیم و داریم:

```
classifiers.bagging(tree.DecisionTreeClassifier(max_depth=20), 10)
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.99      0.99      0.99      2099
         Yes       0.99      0.99      0.99      2099

    accuracy                           0.99      4198
   macro avg       0.99      0.99      0.99      4198
weighted avg       0.99      0.99      0.99      4198

Accuracy:  0.9928537398761315
For Test Predicts
              precision    recall  f1-score   support

          No       0.40      0.35      0.38       423
         Yes       0.71      0.74      0.73       887

    accuracy                           0.62      1310
   macro avg       0.55      0.55      0.55      1310
weighted avg       0.61      0.62      0.61      1310

Accuracy:  0.6183206106870229
```

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.97      0.99      0.98      2099
         Yes       0.99      0.97      0.98      2099

    accuracy                           0.98      4198
   macro avg       0.98      0.98      0.98      4198
weighted avg       0.98      0.98      0.98      4198

Accuracy:  0.98261076703192
For Test Predicts
              precision    recall  f1-score   support

          No       0.48      0.52      0.50       423
         Yes       0.76      0.74      0.75       887

    accuracy                           0.66      1310
   macro avg       0.62      0.63      0.62      1310
weighted avg       0.67      0.66      0.67      1310

Accuracy:  0.6648854961832061
```

همان‌طور که می‌بینیم دقت برای داده‌های ترین نزدیک به ۱ است یعنی مدل به شدت overfit است در این حالت دقت برای تست باید خیلی کم باشد ولی همان‌طور که میبینیم این دقت خیلی کم نشده است یعنی با اینکه مدل overfit است ولی دقت داده‌های تست باز هم زیاد است و مدل توانسته بخشی از این overfit را جلوگیری کند که این از ویزگی های مدل های گروهی است

۴) bootstraping:

روش bootstrappingیک روش آماری برای تخمین ویژگی یک جمعیت با استفاده از میانگین گیری از نتایج مربوط به گروه های کوچک از همان جمعیت است. همچنین این گروه ها ممکن است با هم اشتراک داشته باشند. این موضوع کمک می کند یک مشاهده و اطلاعات چندین بار تکرار شود.
این مراحل را می توان به این صورت خلاصه کرد: انتخاب تعداد samle / برداشتن از جمعیت اصلی/ بازگرداندن sample های انتخاب شده به جمعیت اصلی
این متد برای کم کردن خطای یک مدل machine learning نیز به کار می رود. به این صورت که یک مدل بر اساس داده های تقسیم شده از داده های train آموزش می بیند. بنابراین مدل هایی با ویژگی ها ی متفاوت تولید می شوند و سپس برای داده های unseen از میانگین نتایج این مدل ها استفاده می کنیم.
یکی از ویژگی های مهم استفاده از این روش این است که نتایج توزیع گوسی دارند.
حال تاثیر این روش را بر واریانس و بایاس بررسی می کنیم:
مدلی را مدل biased می نامیم که همیشه پیش بینی را کمتر یا بیشتر از حد واقعی انجام دهد. این مورد می تواند ناشی از انتخاب train data و یا مدلسازی اشتباه ما باشد.
واریانس نیز نشان دهنده تغییرات مدل هنگام اعمال آن بر روی داده های جدید است. در واقع نشان می دهد مدل ما چقدر قابلیت generalization دارد. هر چه واریانس بیشتر باشد مدل ما بد عمل

می کند و overfitting دارد. به صورت کلی ولی مدلی با پیچیدگی و feature های فراوان ممکن است overfitting و واریانس بالایی داشته باشد ولی از طرفی biased نباشد. بنابراین می تواند دقیق تر داده ها را کلاس بندی کند.
از طرفی برای مدل های ساده تر ممکن است دچار واریانس زیاد نشویم ولی مدل ما high biased باشد زیرا نتوانسته است به خوبی الگوی دیتای train را دنبال کند. بنابراین به صورت کلی و برای یک مدل کلی ما دنبال low bias-low variance هستیم.

تکنیک bootstrapping بر این اساس کار می کند که میانگین خطا های داده های مستقل به سمت صفر میل می کند.
به صورت کلی می توان گفت این روش ممکن است گاهی باعث بالا روفتن bias بشود زیرا ما مدل را بر روی داده های کوچک می سازیم و سپس میانگین می گیریم. ولی باعث کاهش واریانس خواهد شد.

حال می‌خواهیم یک مدل دیگر به اسم hard voting را پیاده‌سازی کنیم
این مدل چند الگوریتم را می‌گیرد و با همی آن‌ها داده‌ها را پیش‌بینی می‌کند هر کدام که بهتر بود جواب آخر را می‌دهد
در این پروژه با استفاده از الگوریتم هایی که در فاز اول بهینه کردیم یک مدل hard voting می‌سازیم و نتایج را می‌بینیم
داریم:

```
def voting(self, estimators):
    vote = VotingClassifier(estimators=estimators)
    vote.fit(self.x_train, self.y_train)

    y_predict_train = vote.predict(self.x_train)
    y_predict = vote.predict(self.x_test)

    print("For Train Predicts\n", sklearn.metrics.classification_report(self.y_train, y_predict_train))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_train, y_predict_train))
    print("For Test Predicts\n", sklearn.metrics.classification_report(self.y_test, y_predict))
    print("Accuracy: ", sklearn.metrics.accuracy_score(self.y_test, y_predict))
```

و برای اجرا باید الگوریتم ها را به آن بدهیم:

```
voting([('dt', tree.DecisionTreeClassifier(max_depth=4)), ('knn', neighbors.KNeighborsClassifier(n_neighbors=40)), ('lr', linear_model.LogisticRegression()
```

و بعد از اجرا داریم:

```
For Train Predicts
              precision    recall  f1-score   support

          No       0.70      0.71      0.70      2099
         Yes       0.70      0.70      0.70      2099

    accuracy                           0.70      4198
   macro avg       0.70      0.70      0.70      4198
weighted avg       0.70      0.70      0.70      4198

Accuracy:  0.7024773701762744
For Test Predicts
              precision    recall  f1-score   support

          No       0.50      0.66      0.57       423
         Yes       0.81      0.68      0.74       887

    accuracy                           0.67      1310
   macro avg       0.65      0.67      0.65      1310
weighted avg       0.71      0.67      0.68      1310

Accuracy:  0.6748091603053435
```

می‌بینیم دقت آن چیزی بین ۳ دقت الگوریتم ها است که نزدیک به بهترین آن است و این به دلیل hard vote بودن الگوریتم است چرا که همیشه بهترین جواب انتخاب می‌شود

۶)
مدل های گروهی مثل hard voting باید با الگوریتم های متفاوت و به تعداد زیاد خروجی را حساب کنند ولی در اینجا از ۳ الگوریتم استفاده شده که ۲ تای آن‌ها بسیار شبیه به هم هستند. در اینجا درخت تصمیم و همسایگی در اکثر موارد یک خروجی می‌دهند که در این صورت خروجی hard voting نیسز همان می‌شود زیرا ۲ تا از ۳ تا بوده است برای همین خیلی فرقی با حالت عادی و تکی ندارد و در عین حال طولانی‌تر و پیچیده‌تر شده است
ولی در مواقع ای که الگوریتم ها زیاد تر و با هم فرق دارند می‌تواند دقت را افزایش دهد چرا که اکثریت الگوریتم ها ممکن است درست پیش‌بینی کنند
برای مدل های گروهی دیگر نیز همین قضیه برقرار است