گزارش پروژه درس داده کاوی

دکتر حسین رحمانی

صبا اسماعیلی

۴۰۰۷۲۳۰۰۵

بهار ۱۴۰۰

به دلیل حجم بالای دیتاست نمیتوان همه ی فایل های "csv" را با هم مرج کرد و از یه فایل کلی استفاده کرد.از طرفی قبلانجام کار بر روی دیتا باید دیتای مورد نظر را تمیز کرد . پس در ابتدا ما هر فایل را خوانده و مقادیری که نیاز به حذف شدن دارند یا مقدار null دارند را حذف میکنیم و در فایل جدیدی تحت عنوان clean ذخیره میکنیم.

```
for file_name in glob.glob("dataset/*.csv"):
    name = pathlib.Path(file_name).name.split(".")[0]
    out = "clean" + name + ".csv"
    df = pd.read_csv(file_name, usecols=lambda c: c.strip() not in drop)
    cols_map = {col: col.strip() for col in df.columns}
    df.rename(columns=cols_map, inplace=True)
    df.replace([np.inf, -np.inf], np.nan, inplace=True)
    df.dropna(inplace=True)
    df.to_csv(out,index=False)
```

در مرحله ی بعد دیتاهای تمیز شده را بر اساس label به دو دسته Benign و Attack تقسیم بندی میکنیم و در فایل های جدید ذخیر میکنیم.

```
file_names = glob.glob("dataset/clean/*.csv")
for file_name in file_names:
    name = pathlib.Path(file_name).name.split(".")[0]
    df = pd.read_csv(file_name)
    df[df["Label"] == "BENIGN"].to_csv("Benign " + name + ".csv", index=False)
    df[df["Label"] != "BENIGN"].to_csv("Attack " + name + ".csv", index=False)
```

نمونه برداری از دیتاست باید به گونه ای باشد که balance در کلاس ها بهم نریزد یعنی باید دیتاهامون رو براساس stratified sampling نمونه برداریم.پس به تعداد Benign هایی که وجود دارد از هر کلاس به نسبت دیتاهای هر کلاس به صورت رندوم نمونه برداری میکنیم.

و همه ی کلاس های Benign را خوانده و تعداد Benign را بدست اورده و همه کلاس ها را با هم concat میکنیم .

سپس با توجه به تعداد Benign ها از کلاس attack و با توجه به نسبت داده ها از ان نمونه برداری میکنیم و در کلاس Attack همه را با هم مرج میکنیم.

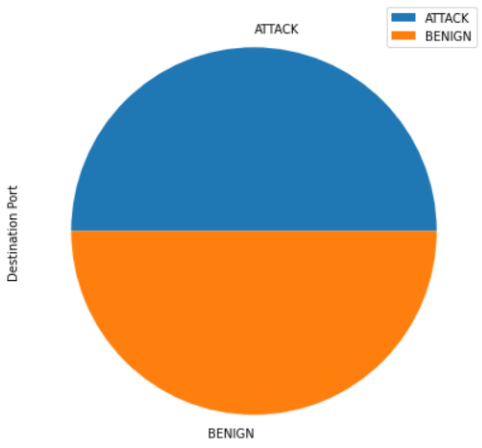
```
benigns = []
for file in sample_percentage.keys():
    name = "Benign clean" + file + ".csv"
    df = pd.read_csv(name)
    benigns.append(df)

benigns = pd.concat(benigns)
benigns.to_csv("Benign.csv", index=False)
benign_size = len(benigns)

attacks = []
for file, p in sample_percentage.items():
    name = "Attack clean" + file + ".csv"
    sample_size = math.floor(p * benign_size)
    df = pd.read_csv(name).sample(sample_size)
    attacks.append(df)

attacks = pd.concat(attacks)
benigns.to_csv("Attack.csv", index=False)
```

در مرحله ی اخر فایل ها را به یک فایل نهایی تبدیل میکنیم که الگوریتم های مورد نیاز را بر روی دیتای نهایی اجرا کنیم.

```
df1 = pd.read_csv("dataset/Attack.csv")
df2 = pd.read_csv("dataset/Benign.csv")
df = pd.concat([df1, df2])
df.to_csv("final.csv", index=False)
```

همانطور که در نمودار pie زیر میبینید دیتاهای نهایی به دو کلاس برابر تقسیم شده است.

```
dataset = pd.read_csv("final.csv")
dataset.groupby("Label").count().plot(figsize=(7,7),kind="pie", y="Destination Port")
```



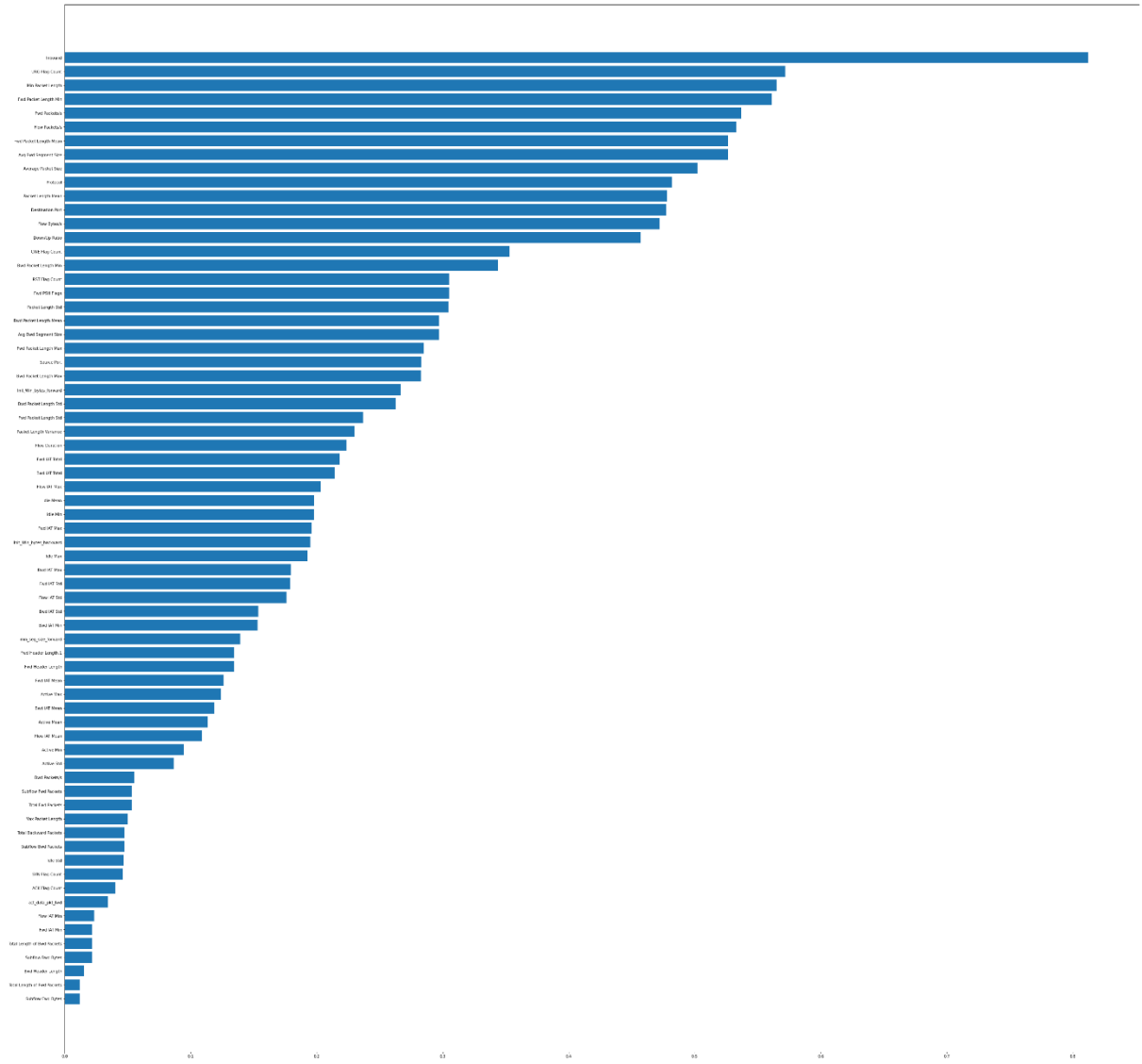
Correlation:

با استفاده از متد pearson برای دیتاست correlation را محاسبه میکنیم.

چون یک سری از داده ها correlation منفی دارند پس از قدر مطلق استفاده میکنیم.

Correlation

```
correlations = dataset.corr(method ='pearson')["Label"].map(abs)
correlations = correlations.drop("Label").values
index = correlations.argsort()
df = dataset.drop("Label", axis=1)
plt.figure(figsize=(50, 50))
plt.barh(df.columns[index], correlations[index])
```



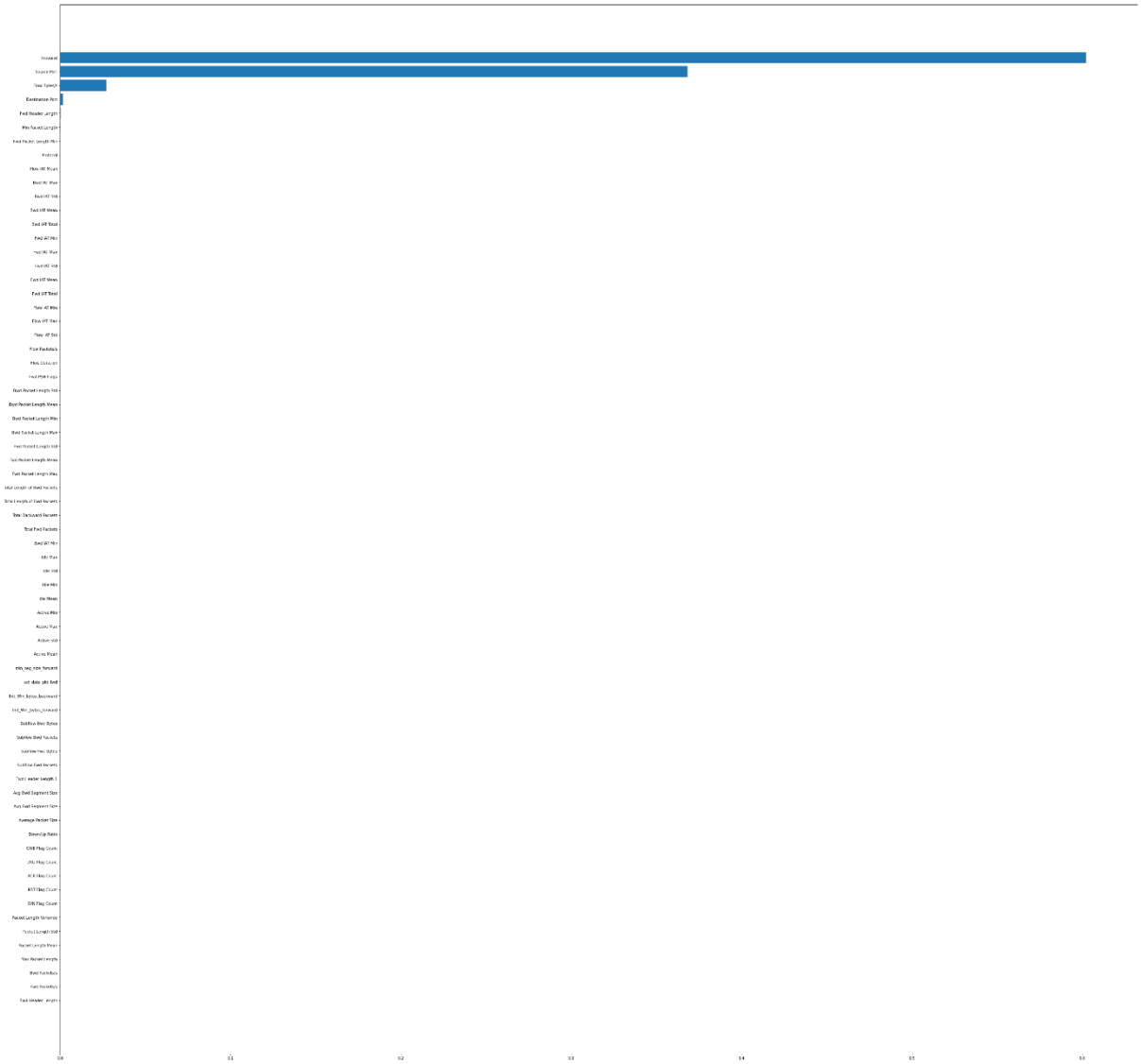
با توجه به نمودار correlation میتوان فهمید متغیر های inbound و urg flag count و min packet length میزان همبستگی بالایی دارند.

Decision Tree:

الگوریتم بعدی برای تحلیل داده ها Decision Tree است.

Decision Tree

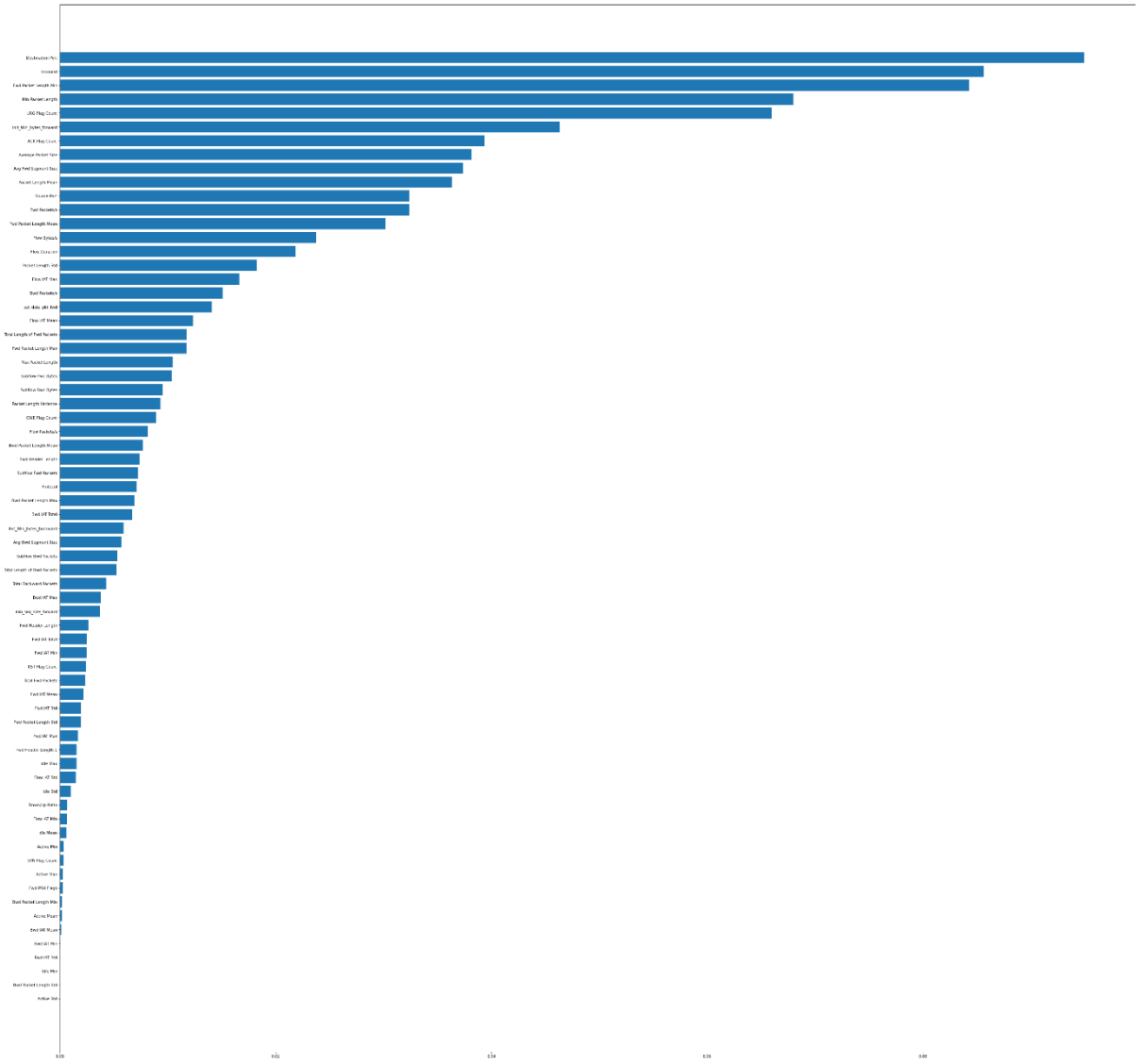
```
dt = DecisionTreeClassifier(criterion="entropy")
dt.fit(xs_df, ys_df)

fi = dt.feature_importances_
index = fi.argsort()
df = dataset.drop("Label", axis=1)
plt.figure(figsize=(50, 50))
plt.barh(xs_df.columns[index], fi[index])
```



با توجه به decision tree میتوان فهمید که فیچر inbound و source port از اهمیت بسیار بالایی برخوردار هستند.

Random Forest:

```
rf = RandomForestClassifier(criterion="entropy")
rf.fit(xs_df, ys_df)

fi = rf.feature_importances_
index = fi.argsort()
df = dataset.drop("Label", axis=1)
plt.figure(figsize=(50, 50))
plt.barh(xs_df.columns[index], fi[index])
```



در الگوریتم random forest فیچر های destination port و inbound و fwd packet length min مقدار بالاتری دارند پس از اهمیت بیشتری برخوردارند.

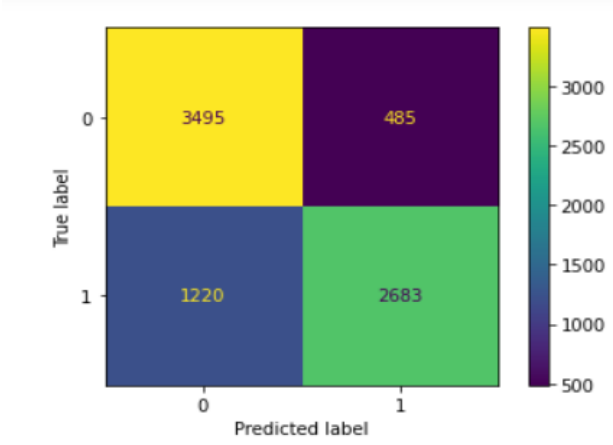
Logistic Regression

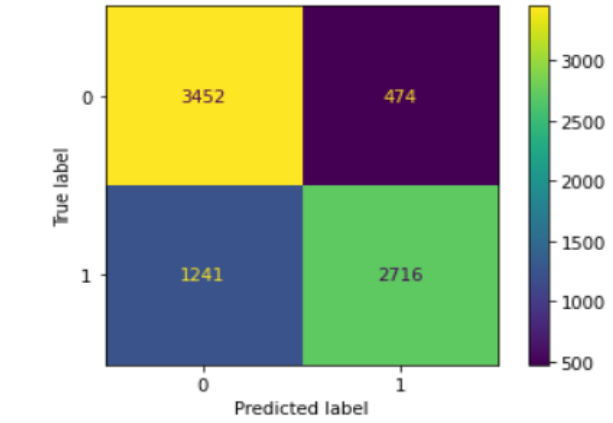
برای test و train داده ها از روش kfold استفاده میکنیم مقدار k را در این پروژه 10 در نظر میگیریم یعنی دیتا را به 10 قسمت مساوی تقسیم میکنیم و هر بار یک fold مدل را ارزیابی میکنیم و با fold های باقی مانده مدل را train میکنیم.

```
xs = normalize(xs_df[variables])
ys = ys_df.values
x_train, x_test, y_train, y_test = train_test_split(xs, ys, test_size=0.3)
kf = KFold(n_splits=10)
```

Logistic Regression

```
for i_train, i_val in kf.split(x_train):
    x_t = x_train[i_train]
    y_t = y_train[i_train]
    x_v = x_train[i_val]
    y_v = y_train[i_val]
    ps = LogisticRegression().fit(x_t, y_t).predict(x_v)

    print("accuracy:", accuracy_score(y_v, ps))
    print("precision:", precision_score(y_v, ps))
    print("recall:", recall_score(y_v, ps))
    print("fscore:", f1_score(y_v, ps))
    print("*"*10)
    ConfusionMatrixDisplay.from_predictions(y_v, ps)
```



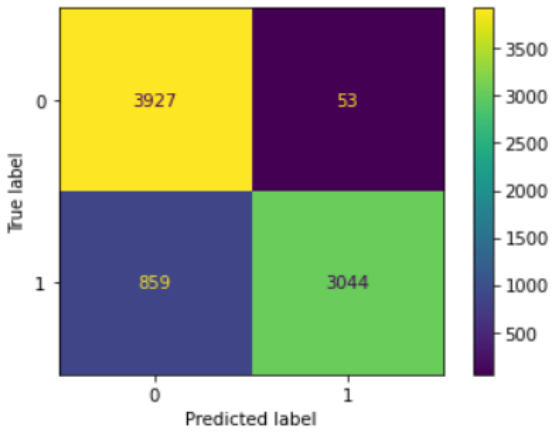


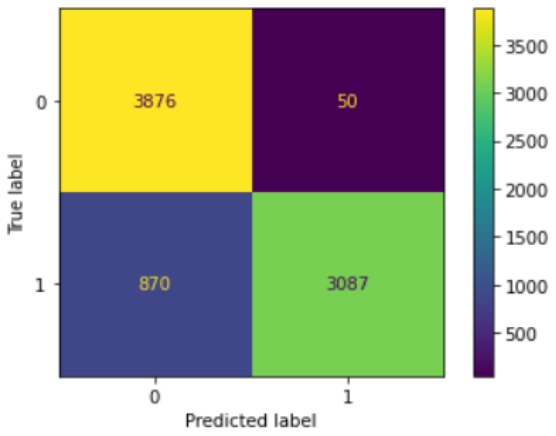

```
accuracy: 0.7837117848534821
precision: 0.8469065656565656
recall: 0.687419933384576
fscore: 0.7588742752085984
**********
accuracy: 0.7824432322719777
precision: 0.8514106583072101
recall: 0.6863785696234521
fscore: 0.7600391772771793
**********
accuracy: 0.7839654953697831
precision: 0.8440024860161591
recall: 0.6933877967832525
fscore: 0.7613174491941135
**********
accuracy: 0.7813015349486236
precision: 0.8499539736115372
recall: 0.6916354556803995
fscore: 0.7626651982378855
**********
accuracy: 0.7824432322719777
precision: 0.8541797611565053
recall: 0.6848072562358276
fscore: 0.7601734023213536
**********
accuracy: 0.7839380867800051
precision: 0.8622222222222222
recall: 0.681555834378921
fscore: 0.7613174491941136
**********
accuracy: 0.7786094899771632
precision: 0.8409163219853643
recall: 0.6797839506172839
fscore: 0.7518133978096999
**********
```

confusion matrix نشان میدهد که این مدل از بقیه مدل ها نتیجه بدتری داشته است و مقدار false positive بسیار بالایی دارد که در پیشبینی ها در یک اپلیکیشن ممکن است باعث بلاک شدن بی دلیل کلاینت ها شود.

Support Vector Machine

```
for i_train, i_val in kf.split(x_train):
    x_t = x_train[i_train]
    y_t = y_train[i_train]
    x_v = x_train[i_val]
    y_v = y_train[i_val]
    ps = SVC(gamma="auto").fit(x_t, y_t).predict(x_v)

    print("accuracy:", accuracy_score(y_v, ps))
    print("precision:", precision_score(y_v, ps))
    print("recall:", recall_score(y_v, ps))
    print("fscore:", f1_score(y_v, ps))
    print("*"*10)
    ConfusionMatrixDisplay.from_predictions(y_v, ps)
```



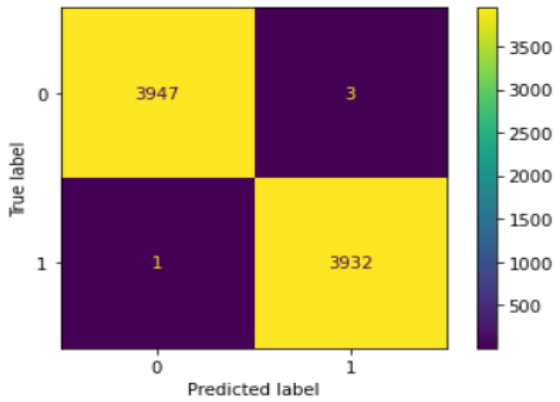


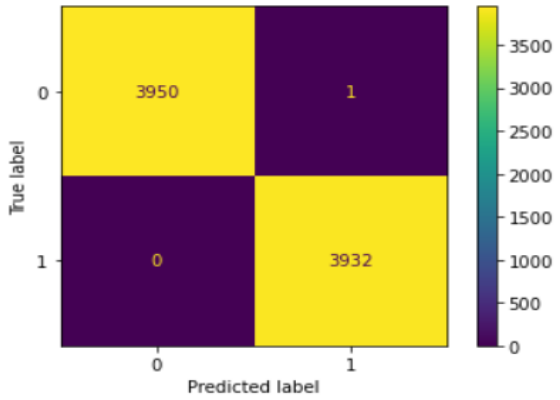

```
accuracy: 0.8843080045667893
precision: 0.9828866645140458
recall: 0.7799128875224186
fscore: 0.8697142857142857
**********
accuracy: 0.8832931625015857
precision: 0.9840612049729041
recall: 0.78013646702047
fscore: 0.8703129405131096
**********
accuracy: 0.8822783204363821
precision: 0.9838135318873422
recall: 0.7758488639264743
fscore: 0.8675421067656296
**********
accuracy: 0.881136623113028
precision: 0.9839116719242902
recall: 0.7787765293383271
fscore: 0.8694076655052265
**********
accuracy: 0.8802486363059748
precision: 0.985553772070626
recall: 0.7734945830183926
fscore: 0.8667419536984754
**********
accuracy: 0.8828977416899264
precision: 0.9878903760356915
recall: 0.7779171894604768
fscore: 0.870419766952127
**********
accuracy: 0.8807409286982999
precision: 0.9845496383957922
recall: 0.7703189300411523
fscore: 0.8643578643578644
**********
```

الگوریتم svm نتیجه بهتری نسبت به الگوریتم قبلی داشته ولی مدت زمان بیشتری برای اجرا نیاز داشت .

Decision Tree

```
for i_train, i_val in kf.split(x_train):
    x_t = x_train[i_train]
    y_t = y_train[i_train]
    x_v = x_train[i_val]
    y_v = y_train[i_val]
    ps = DecisionTreeClassifier(criterion="gini").fit(x_t, y_t).predict(x_v)

    print("accuracy:", accuracy_score(y_v, ps))
    print("precision:", precision_score(y_v, ps))
    print("recall:", recall_score(y_v, ps))
    print("fscore:", f1_score(y_v, ps))
    print("*"*10)
    ConfusionMatrixDisplay.from_predictions(y_v, ps)
```

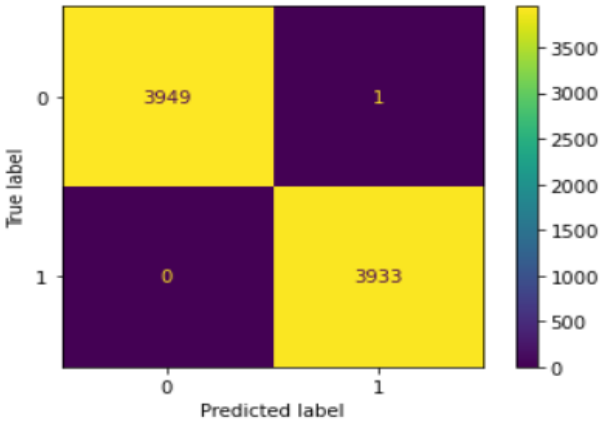


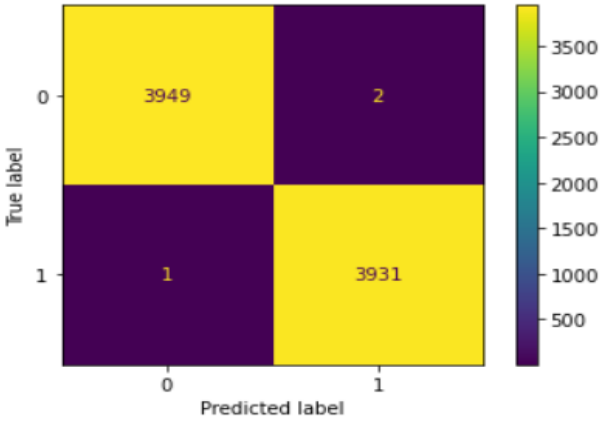



```
accuracy: 0.9994925789673982
precision: 0.9992376111817026
recall: 0.9997457411645054
fscore: 0.9994916115912555
**********
accuracy: 0.9998731447418495
precision: 0.9997457411645054
recall: 1.0
fscore: 0.999872854418309
**********
accuracy: 0.9996194342255487
precision: 0.9994895354772844
recall: 0.999744702578504
fscore: 0.9996171027440971
**********
accuracy: 0.9996194342255487
precision: 0.9994958406856567
recall: 0.9997478567826525
fscore: 0.9996218328501197
**********
accuracy: 0.9998731447418495
precision: 0.9997486173956762
recall: 1.0
fscore: 0.9998742928975487
**********
accuracy: 0.9998731286475514
precision: 0.9997483010319658
recall: 1.0
fscore: 0.9998741346758967
**********
accuracy: 0.9997462572951028
precision: 0.9997493106041614
recall: 0.9997493106041614
fscore: 0.9997493106041614
**********
```

Random Forest

```
for i_train, i_val in kf.split(x_train):
    x_t = x_train[i_train]
    y_t = y_train[i_train]
    x_v = x_train[i_val]
    y_v = y_train[i_val]
    ps = RandomForestClassifier().fit(x_t, y_t).predict(x_v)

    print("accuracy:", accuracy_score(y_v, ps))
    print("precision:", precision_score(y_v, ps))
    print("recall:", recall_score(y_v, ps))
    print("fscore:", f1_score(y_v, ps))
    print("*"*10)
    ConfusionMatrixDisplay.from_predictions(y_v, ps)
```






```
accuracy: 0.9998731447418495
precision: 0.9997458057956279
recall: 1.0
fscore: 0.9998728867420872
**********
accuracy: 0.9996194342255487
precision: 0.999491482329011
recall: 0.9997456765005086
fscore: 0.9996185632549268
**********
accuracy: 0.9994925789673982
precision: 0.9994894051570079
recall: 0.9994894051570079
fscore: 0.9994894051570079
**********
accuracy: 0.9996194342255487
precision: 0.9994958406856567
recall: 0.9997478567826525
fscore: 0.9996218328501197
**********
accuracy: 0.9997462894836991
precision: 0.9997485541865728
recall: 0.9997485541865728
fscore: 0.9997485541865728
**********
accuracy: 0.9997462572951028
precision: 0.9997482376636455
recall: 0.9997482376636455
fscore: 0.9997482376636455
**********
accuracy: 0.9996193859426542
precision: 0.9994987468671679
recall: 0.9997493106041614
fscore: 0.9996240130342148
**********
```

همانطور که میبینیم بهترین الگوریتم random forest است.