به نام خدا

تمرین سری یکم مبانی یادگیری ماشین

معصومه پاسبانی 99243022

سوالات تئوری

سوال اول)

Date: / / Subject:

الف)

$$H(x) = -\sum_{x \in X} p(x) \log_2 p(x)$$

احتمال خرید محصول: $P(yes) = 0.6$

احتمال عدم خرید محصول: $P(NO) = 0.4$

$$H(x) = -\left(\left(P(yes) \log_2(P(yes))\right) + \left(P(NO) \log_2(P(NO))\right)\right)$$

$$= -\left(\left(0.6 \times (-0.736)\right) + \left(0.4 \times (-1.322)\right)\right)$$

$$H(x) = 0.9704$$

ب)

$$H(y|x) = \sum_{x \in X} p(x) H(y|x)$$

عدم دانشجو بودن: $6/10$ ← $P(yes) = 3/6$ $P(NO) = 3/6$

دانشجو بودن: $4/10$ ← $P(yes) = 3/4$ $P(NO) = 1/4$

$$H(x) = 6/10 \left(-3/6 \times \underbrace{\log_2 3/6}_{-1} + -3/6 \times \underbrace{\log_2 3/6}_{-1}\right) +$$

$$4/10 \left(-1/4 \times \underbrace{\log_2 1/4}_{-2} + -3/4 \times \underbrace{\log_2 3/4}_{-0.41}\right) =$$

$$\Rightarrow 0.6 + 0.4(0.5 + 0.3075) = 0.923$$

Scanned with CamScanner

Date: / / Subject:

ب)

$$H(\text{جوان}) = -\left(\tfrac{1}{4}\log_2^{1/4} + \tfrac{3}{4}\log_2^{3/4}\right) \Rightarrow -(-0.5-0.31125) = 0.811$$

$$H(\text{میانسال}) = -\left(\tfrac{2}{2}\log_2^{1} + 0\right) = 0$$

$$H(\text{پیر}) = -\left(\tfrac{3}{4}\log_2^{3/4} + \tfrac{1}{4}\log_2^{1/4}\right) = 0.811$$

$$H(\text{برای سن}) = 0.4(0.811) + 0 + 0.4(0.811) = \underline{0.649}$$

$$IG = H(\text{کل}) - H(\text{سن})$$

$$\Rightarrow 0.9704 - 0.649 = \underline{0.3214}$$

ت)

$$IG = H(\text{کل}) - H(\text{دانشجو بودن}) \Rightarrow 0.0474$$

همانطور که میبینیم IG به دست آمده در مرحله قبل یعنی تقسیم بندی براساس سن، بزرگتر است و این معنی است که عدم اطمینان را کاهش می‌دهد.
در این صورت، موقع تقسیم در اولین تقسیم، عدم قطعیت کمتری خواهد داشت که می‌تواند منجر به یک مدل ساده‌تر و دقیق‌تر شود. زیرا IG بالاتری دارد و نشان دهنده این است که سن، یک ویژگی آموزنده‌تر برای پیش‌بینی رفتار خرید محصول در این dataset است.

Scanned with CamScanner

سوال دوم)

1. مسئله تعداد زیاد ابعاد

در KNN ، فاصله بین نقاط (معمولاً فاصله اقلیدسی) محاسبه می‌شود. با افزایش تعداد ابعاد، فاصله بین نقاط داده‌ها نیز به شدت افزایش می‌یابد، و داده‌ها به نوعی پراکنده‌تر می‌شوند.به طوری که الگوریتم KNN به طور موثر نمی‌تواند بر روی داده‌ها عمل کند و این موضوع باعث می‌شود که تشخیص نزدیکی نقاط داده دشوارتر شود و در نتیجه، دقت مدل کاهش یابد.

در ابعاد زیاد، نقاط داده ممکن است از هم فاصله بگیرند و مدل نمی‌تواند تشخیص دقیقی داشته باشد، زیرا همه نقاط به نظر مشابه خواهند رسید، همچنین حاسبه فاصله بین نقاط نیاز به زمان بیشتری دارد.

برای رفع این مشکل از روش های کاهش ابعاد مانند PCA یا LDA میتوان استفاده کرد تا ویژگی های اصلی داده‌ها حفظ شوند و فضای جستجو کوچکتر شود؛ همچنین انتخاب ویژگی‌های مهم و حذف ویژگی‌های غیرضروری می‌تواند باعث کاهش ابعاد شود و تأثیر کمتری روی دقت مدل بگذارد.

2. مسئله مقیاس بندی ویژگی ها

در KNN ، چون برای پیدا کردن نزدیک‌ترین همسایگان از فاصله‌ها (مثلاً فاصله اقلیدسی) استفاده می‌شود، مقیاس ویژگی‌ها بسیار مهم است. اگر ویژگی‌ها مقیاس‌های متفاوتی داشته باشند، ویژگی‌هایی با مقیاس بزرگتر به طور غیرعادلانه تأثیر بیشتری بر محاسبه فاصله خواهند داشت.

اگر ویژگی‌ها مقیاس‌های متفاوتی داشته باشند، ویژگی‌هایی که دارای مقیاس بزرگتر هستند، می‌توانند تأثیر زیادی در انتخاب نزدیک‌ترین همسایگان داشته باشند؛ همچنین تفاوت مقیاس بین ویژگی‌ها باعث می‌شود که برخی از ویژگی‌ها بیشتر از سایرین در تشخیص کلاس‌ها تأثیر بگذارند.

برای رفع این مشکل میتوان از تکنیک های مقیاس بندی مانند min-max scaling، standardization و normalization استفاده کرد که هر کدام به ترتیب به این صورت عمل میکنند: مقیاس داده ها را به بازه [0,1] تغییر داده، داده‌ها را به گونه‌ای تغییر می‌دهد که میانگین آن‌ها صفر و انحراف معیار یک باشد و داده‌ها را به صورت نسبی و در یک مقیاس خاص استاندارد کنید تا تأثیر ویژگی‌های بزرگتر کاهش یابد.

سوال سوم)

Random Forest الگوریتمی است که از مجموعه‌ای از درخت‌های تصمیم برای بهبود دقت و پایداری پیش‌بینی‌ها استفاده می‌کند.

ابتدا برای ساخت هر درخت در جنگل تصادفی، نمونه‌ای از داده‌های آموزشی به‌طور تصادفی با جایگزینی انتخاب می‌شود، یعنی امکان دارد داده‌ای چندین بار انتخاب شود. سپس برای هر درخت، در هر گره تنها از زیرمجموعه‌ای از ویژگی‌ها به‌طور تصادفی استفاده می‌شود. این کار از ایجاد درخت‌هایی که بیش از حد به یک ویژگی خاص وابسته هستند جلوگیری می‌کند و در نهایت در طبقه بندی با رای گیری label ای که بیشترین تعداد درخت‌ها آن را پیش‌بینی کرده‌اند به عنوان خروجی نهایی انتخاب می‌شود و در رگرسیون، میانگین پیش‌بینی‌ها از تمام درخت‌ها به عنوان خروجی نهایی در نظر گرفته می‌شود.

مزایای Random Forest نسبت به درخت تصمیم:

Random Forest به دلیل ترکیب چندین درخت، دقت بالاتری نسبت به درخت تصمیم منفرد دارد. این ترکیب باعث می‌شود که ضعف‌های درخت‌های منفرد کاهش پیدا کند و نتایج نهایی بهینه‌تر باشند. درخت‌های تصمیم به دلیل ساختارشان مستعد overfitting هستند، به‌ویژه زمانی که درخت عمیق و پیچیده می‌شود. در Random Forest، با ترکیب چندین درخت و ایجاد تنوع در آن‌ها، این مشکل کاهش می‌یابد و مدل به داده‌های آموزشی بیش از حد وابسته نمی‌شود. به دلیل اینکه پیش‌بینی نهایی براساس رأی‌گیری یا میانگین چندین درخت به دست می‌آید، Random Forest نسبت به نویز موجود در داده‌ها مقاوم‌تر است و عملکرد پایدارتری دارد.

Random Forest نه‌تنها در طبقه‌بندی بلکه در رگرسیون نیز به خوبی عمل می‌کند و یک روش چندمنظوره است.

سوال چهارم)

Weighted KNN: در الگوریتم استانداردKNN ، کلاس‌بندی بر اساس رأی‌گیری از نزدیک‌ترین همسایگان انجام می‌شود. هر همسایه به طور مساوی اهمیت دارد و تعداد همسایگان هر کلاس تعیین می‌کند که نمونه به کدام کلاس تخصیص داده شود. اما در Weighted KNN، نزدیک‌ترین همسایگان با توجه به فاصله‌شان از نقطه موردنظر وزن‌دهی می‌شوند، به‌طوری که هرچه همسایه به نقطه نزدیک‌تر باشد، وزن بالاتری دارد.

این الگوریتم ابتدا فاصله بین نمونه جدید و تمام نمونه های مجود در مجموعه داده را محاسبه می کند، سپس نزدیک ترین k همسایه را انتخاب می کند. برای هر همسایه، weight = 1/distance محسابه می شود و در نهایت کلاس نمونه جدید با رای گیری وزنی تعیین می شود.

مزایا:

همسایه‌های نزدیک‌تر که اهمیت بیشتری در طبقه‌بندی دارند، تأثیر بیشتری روی پیش‌بینی نهایی دارند، که باعث افزایش دقت طبقه‌بندی می‌شود. این روش به داده‌های توزیع غیریکسان و با فواصل مختلف حساس‌تر است و نسبت به KNN ساده که تمامی همسایگان را به یک اندازه در نظر می‌گیرد، انعطاف‌پذیری بهتری دارد.

NCA: الگوریتم NCAیک روش یادگیری بر اساس متریک است که به منظور افزایش دقت الگوریتم KNN طراحی شده است. در این روش، از بهینه‌سازی یک تابع هدف استفاده می‌شود که هدف آن یادگیری یک نگاشت خطی برای داده‌ها است. این نگاشت خطی، داده‌ها را به شکلی تغییر می‌دهد که همسایگان از نظر فاصله در یک فضای جدید بهینه‌سازی شوند.

NCA سعی می‌کند یک ماتریس تبدیل بیابد که داده‌ها را به یک فضای جدید منتقل کند به طوری که همسایگان مربوط به هر نمونه در این فضای جدید، طبقه‌بندی دقیقی داشته باشند. تابع هزینه‌ای که در این روش بهینه‌سازی می‌شود، احتمال پیش‌بینی درست کلاس توسط نزدیک‌ترین همسایگان را اندازه‌گیری می‌کند. این احتمال بر اساس فاصله‌ها محاسبه می‌شود و NCA سعی دارد که فاصله نمونه‌ها را در این فضای جدید به‌گونه‌ای تغییر دهد که همسایگان نزدیک از همان کلاس باشند. با استفاده از روش‌های بهینه‌سازی، این ماتریس نگاشت خطی به دست می‌آید، سپس داده‌ها به این فضای جدید تبدیل می‌شوند و الگوریتم KNN روی داده‌های تبدیل شده اعمال می‌شود.

مزایا:

با یادگیری نگاشتی که داده‌ها را به فضای جدیدی منتقل می‌کند، الگوریتم KNN در این فضا دقت بالاتری دارد. این روش به خصوص در مسائل پیچیده و داده‌های با توزیع غیرهمگن عملکرد بهتری دارد، زیرا ساختار داده‌ها را بهینه‌تر تنظیم می‌کند.

سوالات عملی

سوال اول)

```
[55] X = data.iloc[:, :-1].values
     y = data.iloc[:, -1].values

     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[56] depths = range(1, 70)
     depth_acc_dt, depth_prec_dt, depth_rec_dt = [], [], []
     depth_acc_rf, depth_prec_rf, depth_rec_rf = [], [], []

     max_features = range(1, X.shape[1] + 1)
     feature_acc_dt, feature_prec_dt, feature_rec_dt = [], [], []
     feature_acc_rf, feature_prec_rf, feature_rec_rf = [], [], []
```

ابتدا داده ها را به دو بخش train و test تقسیم کرده که این تقسیم‌بندی برای ارزیابی عملکرد مدل‌ها بر روی داده‌ای که در مرحله آموزش دیده نشده‌اند، استفاده می‌شود.

```
     for depth in depths:
57]      # Decision Tree
         dt = DecisionTreeClassifier(max_depth=depth)
         dt.fit(X_train, y_train)
         y_pred_dt = dt.predict(X_test)
         depth_acc_dt.append(accuracy_score(y_test, y_pred_dt))

         # Random Forest
         rf = RandomForestClassifier(max_depth=depth)
         rf.fit(X_train, y_train)
         y_pred_rf = rf.predict(X_test)
         depth_acc_rf.append(accuracy_score(y_test, y_pred_rf))
```

```
58] for depth in depths:
         # Decision Tree
         dt = DecisionTreeClassifier(max_depth=depth)
         dt.fit(X_train, y_train)
         y_pred_dt = dt.predict(X_test)
         depth_prec_dt.append(precision_score(y_test, y_pred_dt))

         # Random Forest
         rf = RandomForestClassifier(max_depth=depth)
         rf.fit(X_train, y_train)
         y_pred_rf = rf.predict(X_test)
         depth_prec_rf.append(precision_score(y_test, y_pred_rf))
```

سپس مدل‌های درخت تصمیم و جنگل تصادفی با مقادیر مختلف max_depth آموزش داده می‌شوند. این پارامتر تعیین می‌کند که حداکثر عمق درخت چه مقدار باشد و تاثیر مستقیمی بر صحت، precision و recall مدل‌ها دارد. برای هر مقدار از max_depth، سه متریک زیر محاسبه می‌شود:

(Accuracy): نسبت نمونه‌های درست پیش‌بینی شده به کل نمونه‌ها.

(Precision): نسبت نمونه‌های مثبت صحیح پیش‌بینی شده به کل نمونه‌های مثبت پیش‌بینی شده.

(Recall): نسبت نمونه‌های مثبت صحیح پیش‌بینی شده به کل نمونه‌های مثبت واقعی.

```
for depth in depths:
    # Decision Tree
    dt = DecisionTreeClassifier(max_depth=depth)
    dt.fit(X_train, y_train)
    y_pred_dt = dt.predict(X_test)
    depth_rec_dt.append(recall_score(y_test, y_pred_dt))

    # Random Forest
    rf = RandomForestClassifier(max_depth=depth)
    rf.fit(X_train, y_train)
    y_pred_rf = rf.predict(X_test)
    depth_rec_rf.append(recall_score(y_test, y_pred_rf))
```

```
for feature in max_features:
    # Decision Tree
    dt = DecisionTreeClassifier(max_features=feature)
    dt.fit(X_train, y_train)
    y_pred_dt = dt.predict(X_test)
    feature_acc_dt.append(accuracy_score(y_test, y_pred_dt))

    # Random Forest
    rf = RandomForestClassifier(max_features=feature)
    rf.fit(X_train, y_train)
    y_pred_rf = rf.predict(X_test)
    feature_acc_rf.append(accuracy_score(y_test, y_pred_rf))
```

```
[61] for feature in max_features:
         # Decision Tree
         dt = DecisionTreeClassifier(max_features=feature)
         dt.fit(X_train, y_train)
         y_pred_dt = dt.predict(X_test)
         feature_prec_dt.append(precision_score(y_test, y_pred_

         # Random Forest
         rf = RandomForestClassifier(max_features=feature)
         rf.fit(X_train, y_train)
         y_pred_rf = rf.predict(X_test)
         feature_prec_rf.append(precision_score(y_test, y_pred_
```

```
[62] for feature in max_features:
         # Decision Tree
         dt = DecisionTreeClassifier(max_features=feature)
         dt.fit(X_train, y_train)
         y_pred_dt = dt.predict(X_test)
         feature_rec_dt.append(recall_score(y_test, y_pred_dt))

         # Random Forest
         rf = RandomForestClassifier(max_features=feature)
         rf.fit(X_train, y_train)
         y_pred_rf = rf.predict(X_test)
         feature_rec_rf.append(recall_score(y_test, y_pred_rf))
```

s

پارامتر max_features تعداد ویژگی‌هایی که مدل در هر بار انشعاب درخت استفاده می‌کند را تعیین می‌کند. این بخش مشابه قسمت قبل است، اما پارامتر max_features به جای max_depth تغییر می‌کند. برای هر مقدار از max_features، صحت precision و recall محاسبه می‌شوند.

سپس نمودارها را برای تحلیل تغییرات صحت، دقت و فراخوانی نسبت به max_depth و max_features رسم کرده. این نمودارها کمک می‌کنند که تاثیر این پارامترها بر عملکرد مدل‌ها را بهتر مشاهده و مقایسه کنیم. هر نمودار شامل خطی برای مدل درخت تصمیم و خط دیگری برای مدل جنگل تصادفی است.

سوال دوم)

توازن‌سازی با استفاده از نمونه‌برداری مجدد Oversampling وUndersampling

Oversampling :در این روش، داده‌های کلاس کم‌نمونه چندین بار تکرار می‌شوند یا نمونه‌های مصنوعی برای این کلاس ایجاد می‌شود تا اندازه کلاس‌های مختلف به یکدیگر نزدیک شود. یکی از روش‌های پرکاربرد در این دسته SMOTE (Synthetic Minority Over-sampling Technique)است که با تولید نمونه‌های مصنوعی جدید برای کلاس‌های کم‌نمونه، توازن ایجاد می‌کند.

Undersampling: در این روش، تعداد نمونه‌های کلاس غالب کاهش می‌یابد تا اندازه کلاس‌ها متعادل شود. به عنوان مثال، می‌توان تعدادی از نمونه‌های کلاس غالب را به‌صورت تصادفی حذف کرد.

Cost sensitive learning

این روش به جای متعادل کردن داده‌ها، الگوریتم یادگیری را اصلاح می‌کند تا طبقه‌بندی نادرست کلاس اقلیت را به شدت جریمه کند. در طبقه‌بندی‌کننده‌هایی مانند درخت‌های تصمیم، ماشین‌های بردار پشتیبان یا شبکه‌های عصبی، می‌توان هزینه های بالاتری را برای طبقه‌بندی اشتباه نمونه‌های کلاس اقلیت، بهبود عملکرد برای داده‌های نامتعادل تعیین کرد.

Ensemble techniques

تکنیک‌هایی مانند جنگل تصادفی متعادل یا EasyEnsemble روش‌های مجموعه را با استراتژی‌های نمونه‌گیری ترکیب می‌کنند. برای مثال EasyEnsemble چندین مدل را در مجموعه داده های متعادل با کم نمونه برداری از کلاس اکثریت برای هر مدل آموزش می دهد. سپس نتایج تجمیع می‌شوند و عملکرد در طبقات اقلیت را افزایش می‌دهند.

```
7] le = LabelEncoder()
   values = ['gender', 'ever_married', 'work_type' , 'Residence_type', 'smoking_status']
   for col in values:
     data[col] = le.fit_transform(data[col])
   data.head()
```

ابتدا ستون های غیرعددی با استفاده از labelencoder به داده های عددی تبدیل میشوند. سپس با ایتفاده از simpleImputer داده های میس شده در ویژگی های عددی با میانگین گیری پر میشوند.

```
[8] from sklearn.impute import SimpleImputer
    x = data.drop('stroke', axis=1)
    imputer = SimpleImputer(strategy='mean')
    x = pd.DataFrame(imputer.fit_transform(x), columns=x.columns, index=x.index) #convert b
    y = data['stroke']
```

Double-click (or enter) to edit

```
[9] smote = SMOTE(random_state=42)
    x_balanced, y_balanced = smote.fit_resample(x, y)
    print("Balanced Data Shape:", x_balanced.shape)
    print("Class Distribution After SMOTE:", Counter(y_balanced))

    Balanced Data Shape: (85234, 11)
    Class Distribution After SMOTE: Counter({0: 42617, 1: 42617})

[10] x_train, x_test, y_train, y_test = train_test_split(x_balanced, y_balanced, test_size=0

[11] print("Train Set Shape:", x_train.shape)
     print("Test Set Shape:", x_test.shape)
```

از تکنیک SMOTE برای متعادل‌سازی کلاس‌ها استفاده شده است. این تکنیک نمونه‌های جدیدی را برای کلاس‌های کم‌تعداد ایجاد می‌کند تا توزیع داده‌ها متعادل شود. پس از اعمالSMOTE ، توزیع کلاس‌ها چاپ می‌شود تا مطمئن شویم داده‌ها به‌درستی متعادل شده‌اند

داده‌های متعادل به نسبت ٪۶۷ و ٪۳۳ به مجموعه‌های آموزش و تست تقسیم می‌شوند. این تقسیم‌بندی کمک می‌کند تا از بخشی از داده‌ها برای آموزش مدل استفاده کرده و عملکرد مدل را روی بخشی دیگر ارزیابی کنیم.

```python
def knn(x_train, y_train, x_test, k=9):

    x_train_np = x_train.values
    x_test_np = x_test.values
    y_train_np = y_train.values

    max_labels = []

    for test_row in x_test_np:
        distances = np.linalg.norm(x_train_np - test_row, axis=1)
        k_indices = np.argsort(distances)[:k]
        k_nearest_labels = y_train_np[k_indices]

        most_common_label = Counter(k_nearest_labels).most_common(1)[0][0]
        max_labels.append(most_common_label)

    return np.array(max_labels)
```

```python
prediction = knn(x_train, y_train, x_test)
```

```python
print(classification_report(y_test, prediction))
```

```
              precision    recall  f1-score   support

           0       0.94      0.77      0.85     14074
           1       0.81      0.95      0.87     14054

    accuracy                           0.86     28128
   macro avg       0.87      0.86      0.86     28128
weighted avg       0.87      0.86      0.86     28128
```

```python
accuracy = accuracy_score(y_test, prediction)
accuracy
```

```
0.860494880546075
```

سپس تابع knn را به صورت دستی پیاده سازی میکنیم. برای هر نمونه در مجموعه تست، فاصله اقلیدسی آن نسبت به تمام نمونه‌های مجموعه آموزش محاسبه شده و k نزدیک‌ترین همسایه‌ها انتخاب می‌شوند. لیبل رایج‌ترین همسایه‌ها به‌عنوان پیش‌بینی آن نمونه تست انتخاب می‌شود. پس از پیش‌بینی کلاس‌ها، عملکرد مدل با استفاده از classification_report و accuracy_scoreارزیابی می‌شود.

```
] from sklearn.preprocessing import StandardScaler, PolynomialFeatures

  scaler = StandardScaler()
  x_train_scaled = scaler.fit_transform(x_train)
  x_test_scaled = scaler.transform(x_test)

] poly = PolynomialFeatures(degree=2, interaction_only=True, include_bias=False)
  x_train_poly = poly.fit_transform(x_train_scaled)
  x_test_poly = poly.transform(x_test_scaled)
```

داده‌ها با استفاده از StandardScaler استانداردسازی می‌شوند تا ویژگی‌ها میانگین صفر و واریانس یک داشته باشند.

ویژگی‌های جدید چندجمله‌ای (پلی‌نومیال) با درجه ۲ با استفاده از PolynomialFeatures ایجاد می‌شوند، که شامل اثرات تعاملی بین ویژگی‌ها می‌باشد.

```
metrics = ['euclidean', 'manhattan']
best_k = None
best_score = 0
best_metric = None

for metric in metrics:
    for k in range(1,15):
        knn_model = KNeighborsClassifier(n_neighbors=k, metric=metric)
        knn_model.fit(x_train, y_train)
        prediction = knn_model.predict(x_test)
        score = accuracy_score(y_test, prediction)

        if score > best_score:
            best_score = score
            best_k = k
            best_metric = metric

        print(f"Metric: {metric}, k: {k}, Accuracy: {score:.4f}")

Metric: euclidean, k: 1, Accuracy: 0.9824
Metric: euclidean, k: 2, Accuracy: 0.9817
Metric: euclidean, k: 3, Accuracy: 0.9688
Metric: euclidean, k: 4, Accuracy: 0.9695
Metric: euclidean, k: 5, Accuracy: 0.9585
Metric: euclidean, k: 6, Accuracy: 0.9599
```

از KNeighborsClassifier برای پیاده‌سازی الگوریتم KNN استفاده کرده. این مدل از scikit-learn قابلیت استفاده از متریک‌های فاصله مختلف (مانند اقلیدسی و منهتن) را دارد. مدل با دو متریک فاصله مختلف (اقلیدسی و منهتن) و مقادیر

مختلف k از ۱ تا ۱۴ آزمایش می‌شود تا بهترین ترکیب k و متریک فاصله پیدا شود. نتایج برای هر ترکیب k و متریک فاصله چاپ می‌شوند و در نهایت بهترین مقدار k و متریک فاصله با بالاترین دقت گزارش می‌شوند.