گزارش پروژهٔ هوش مصنوعی – محمدپویا تراشی – ۴۰۱۵۲۱۱۰۲

پیاده‌سازی KNN:

```
class KNNClassifier:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = np.array(y)

    def predict(self, X):
        predictions = []
        for x in X:
            distances = np.sqrt(np.sum((self.X_train - x)**2, axis=1))    # calculate euclid. dist.

            # get k nearest neighbors
            k_indices = np.argsort(distances)[:self.k]
            k_nearest_labels = self.y_train[k_indices]

            prediction = np.bincount(k_nearest_labels).argmax()
            predictions.append(prediction)
        return np.array(predictions)

```

در تابع fit، داده‌های تست در مدل ذخیره می‌شوند. برای انجام پیش‌بینی توسط مدل، فاصلهٔ نقطهٔ مد نظر تا تمامی نقاط موجود در داده‌های train محاسبه می‌گردد و K دادهٔ نزدیک به آن بررسی می‌شوند و بیشترین لیبلی تکراری به عنوان نتیجه گزارش می‌شود.

پیش‌پردازش و آماده‌سازی داده‌ها:

```
def load_and_preprocess_data(path):
    df = pd.read_csv(path)
    cols = ["grade", "term", "home_ownership", "emp_length", "bad_loans"]
    df = df[cols].copy()    # use .copy method to prevent pd to place a view instead of a complete dataframe

    min_class_count = min(df['bad_loans'].value_counts())    # 1 is less, use it as the count of samples
    df_balanced = pd.concat([
        df[df['bad_loans'] == 0].sample(min_class_count),
        df[df['bad_loans'] == 1].sample(min_class_count)
    ])
    # df_balanced now have the same amount of records with values 0 and 1

    # print(df_balanced.head())

    le = LabelEncoder()
    for col in ['grade', 'term', 'home_ownership', 'emp_length']:
        df_balanced[col] = le.fit_transform(df_balanced[col].astype(str))

    # print(df_balanced.head())

    X = df_balanced.drop('bad_loans', axis=1)
    y = df_balanced['bad_loans']

    X_train, X_non_train, y_train, y_non_train = train_test_split(X, y, test_size=0.3, random_state=42)    # leaves 30% of data for test and valid.
    X_val, X_test, y_val, y_test = train_test_split(X_non_train, y_non_train, test_size=0.5, random_state=42)    # 15% for valid. and 15% test


```

```
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)    # calculate mean and variance here to use later for valid. and test
    X_val_scaled = scaler.transform(X_val)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_val_scaled, X_test_scaled, y_train, y_val, y_test, X.columns

```

در این بخش ابتدا ستون‌های مورد نیاز جدا می‌شوند و بقیه drop می‌شوند. در ادامه، به تعداد مساوی sample از انواع bad_loanها گرفته می‌شود. (به تعداد لیبلی که تعداد کمتری دارد.) سپس برای متغیرهای کیفی یک لیبل عددی در نظر گرفته می‌شود.

برای فیچرهای موجود، ستون bad_loans drop می‌شود و مقدار بدست آمده در X ریخته می‌شود. همچنین خود این ستون بعنوان target در متغیر y نگهداری می‌شود.

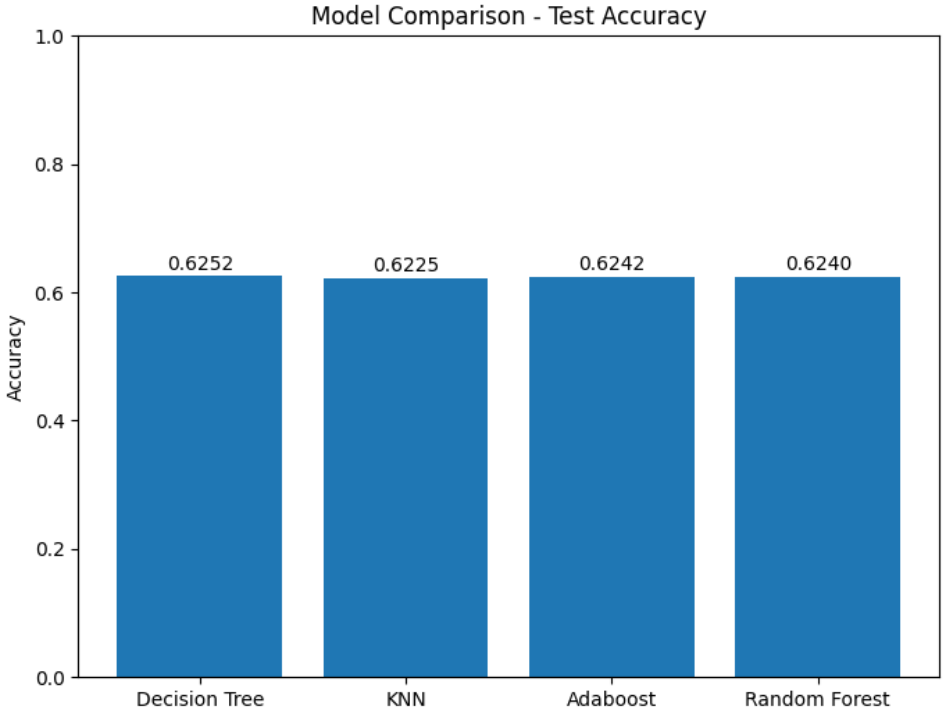
۷۰ درصد از دادهٔ موجود به عنوان دادهٔ آموزشی در نظر گرفته می‌شود. نیمی از دادهٔ باقی‌مانده برای تست و نیم دیگر برای VALIDATION به کار می‌رود.

سپس با استفاده از رابطهٔ $z = \frac{x-\mu}{\sigma}$ داده‌ها SCALE می‌شوند تا الگوریتم‌ها نتیجهٔ منطقی‌تری ارائه دهند. (داده‌های VALIDATION و تست نیز با میانگین و انحراف معیار داده‌های آموزشی SCALE می‌شوند.)

TUNE نمودن ابرپارامترها

برای بهینه‌سازی الگوریتم‌های KNN و درخت تصمیم از یک لیست از مقادیری که به تجربه PROMISING به نظر می‌رسند استفاده گردید. برای الگوریتم‌های جنگل تصادفی و ADABOOST نیز از GRIDSEARCH استفاده شد و بر اساس دقت، این ابرپارامترها محاسبه شدند.

نمودار دقت مدل‌ها



مقایسهٔ الگوریتم‌ها از نظر پیچیدگی زمانی:

	زمان یادگیری	زمان پیش‌بینی
KNN	**O(1) – (LAZY)**	**O(N * D)**
DT	**O(N * D * LOGN)**	**O(LOGN)**
RF	**O(T * N * D * LOGN)**	**O(T * LOGN)**
ADABOOST	**O(T * N * D)**	**O(T * D)**